Sfruttare tutta la potenza
della nuova versione
di Visual Basic:
oltre 800 pagine
di consigli degli esperti

Svelare i misteri
della creazione
di applicazioni Windows
e Web professionali

Nel CD-ROM
programmi
di Blue
Sky Software,
Desaware,
InstallShield,
Sax Software
e VideoSoft

i segreti di Visual Basic 6

Harold Davis

APOGEO
IDG BOOKS WORLDWIDE

Nella stessa collana:

- **I segreti di Windows 98,** di Livingston e Straub
- **I segreti della programmazione in Windows 98,** di Walnum
- **I segreti di Excel 97,** di Barrows
- **I segreti di Visual Basic 5,** di Davis
- **I segreti di AutoCAD 13,** di Walsh, Knight e Valaski
- **I segreti di UNIX,** di Armstrong
- **I segreti di Windows** NT **Server 4.0,** di Hilley
- **I segreti di Linux,** di Barkakati
- **I segreti di Excel per Windows 95,** di Burns e Nicholson
- **I segreti di Word per Windows 95,** di Lowe
- **I segreti del World Wide Web,** di Perry
- **I segreti di Windows 95,** di Livingston e Straub

Volumi di interesse in altre collane:

- **Visual Basic 6 - Guida completa,** di Perry
- **Visual C++ 6 - Guida completa,** di Chapman
- **Visual Basic 6 For Dummies,** di Wang
- **Visual Basic 6 For Dummies Espresso,** di Shammas
- **Visual C++ 6 For Dummies,** di Hyman e Arnson
- **Visual C++ 6 For Dummies Espresso,** di Wright
- **L'API di Windows 98 For Dummies Espresso,** di Shammas e Noonan
- **Programmare con Delphi 4, di Cantù**

HAROLD DAVIS

I segreti di
Visual Basic 6

I segreti di Visual Basic 6

Titolo originale:
Visual Basic 6 Secrets

Autore:
Harold Davis

Viale Papiniano 38 - 20123 Milano (Italy)
Telefono: 02-461920 (5 linee r.a.) - Telefax: 02-4815382
Email apogeo@apogeonline.com
U.R.L. http://www.apogeonli ne.com

ISBN 88-7303-463-2

Collana diretta da Virginio 6. Sala

Realizzazione editoriale di SCRIPT, Cinisello Balsamo (Milano)

Traduzione di Stefano Albarelli, Stefano Brentegani, Carlo Milanesi, Alessandro Muselli, Daniela Parola, Claudio Persuati e Mauro Santabarbara

Impaginazione elettronica di Monica Maltarole

Copertina di Enrico Marcandalli

SOMMARIO

CAPITOLO 5 - CARATTERISTICHE DI LIVELLO AVANZATO ... 111

PARTE SECONDA - PROGRAMMAZIONE WINDOWS

CAPITOLO 6 - INTRODUZIONE AI SISTEMI OPERATIVI ... 125

CAPITOLO 7 - FINESTRE DI DIALOGO COMUNI DI WNNDOWS,,............143

CAPITOLO 8 - CONTROLLI D'INTERFACCIA UTENTE 159

CAPITOLO 9 - USO DEL REGISTRO DI CONFIGURAZIONE.............................201

CAPITOLO 10 - PROGRAMMAZIONE DEL REGISTRO...213

CAPITOLO 11 - VISUAL STUDIO API WIN32 E MESSAGGI..251

CAPITOLO 12 - VISUAL SOURCESAFE (ENTERPRISE EDITION) 301

PARTE TERZA - SEGRETI DI PROGRAMMAZIONE

CAPITOLO 13 - UNA BUONA PRATICA DI PROGRAMMAZIONE 317

CAPITOLO 14 - VISUAL BASIC E L'OOP 343

PARTE QUARTA - SEGRETI DI VISUALIZZAZIONE

CAPITOLO 17 - PROGETTAZIONE DI UNA BUONA INTERFACCIA 431

CAPITOLO 18 - APPLICAZIONI MDI E MENU 441

PARTE V - USO DI ACTIVEX

CAPITOLO 19 - VISUALIZZAZIONE DURANTE L'ESECUZIONE 469

CAPITOLO 20 - CAPIRE ACTIVEX E OLE......................491

CAPITOLO 21 - APPLICAZIONI CHE SUPPORTANO OLE......................509

CAPITOLO 22 - CONTROLLO DI OGGETTI DI APPLICAZIONI ESTERNE 535

CAPITOLO 23 - CREAZIONE DI APPLICAZIONI ACTIVEX 563

PARTE SESTA - CREAZIONE DI CONTROLLI ACTIVEX

CAPITOLO 26 - LE FUNZIONALITÀ DEL CONTROLLO 655

CAPITOLO 27 - CONTROLLI ACTIVEX INSTALLATI VIA WEB 681

PARTE SETTIMA - ESTENSIONE DELL'AMBIENTE

CAPITOLO 28 - APPLICAZIONI INTERNET 699

CAPITOLO 29 - CREAZIONE DI UN ADD-IN PER VISUAL BASIC ... 719

CAPITOLO 30 - COSTRUZIONE DI UN WIZARD ... 749

INTRODUZIONE

Visual Basic è il linguaggio di programmazione più venduto di tutti i tempi e vanta un numero di sviluppatori superiore rispetto a qualsiasi altro ambiente di sviluppo. Con la versione 5 di Visual Basic è stata aggiunta al prodotto una serie di caratteristiche necessarie agli sviluppatori professionali, come la possibilità di creare veri eseguibili compilati e controlli ActiveX. Perché allora tutto questo chiasso per Visual Basic versione 6, l'ultima versione e la migliore?
Tra le caratteristiche di Visual Basic 6 troviamo miglioramenti all'IDE (Integrated Development Environment) e nuove funzionalità, come gli strumenti per Internet e il Web, accesso ai dati facilitato e numerose procedure guidate (Wizard). Ogni nuovo aspetto sarà esaminato con cura nel corso del libro, ma se si desidera poter dare uno sguardo d'insieme a tutte le novità si faccia riferimento al Capitolo 5, che riepiloga tutte le caratteristiche di livello avanzato.
Oltre ai cambiamenti e ai miglioramenti, Visual Basic 6 modifica le condizioni operative degli sviluppatori professionisti. Gli aspetti più significativi in proposito sono:

- Integrazione con altri prodotti di sviluppo Microsoft, compreso Visual Studio e MSDN (Microsoft Developer Network).
- Gestione migliorata dell'accesso ai dati, con il conseguente uso facilitato di Visual Basic per creare applicazioni multilivello che comportano l'accesso a database.
- Funzionalità per Internet che permettono l'uso di Visual Basic allo scopo di creare applicazioni Web.

Scopo di questo libro

Visual Basic nasce come linguaggio "di base", ma attorno al modesto nucleo iniziale è cresciuto un vasto sistema di estensioni, tecniche e strumenti di terze parti. Senza dubbio è facile usare Visual Basic per fare cose semplici, ma sfruttare tutte le sue potenzialità è molto più difficile. Con il passare del tempo Visual Basic è diventato sempre più elaborato e potente ed è quindi sorta l'esigenza di un libro che rivelasse i trucchi del mestiere dal punto di vista dello sviluppatore.

La programmazione, quando è ottimale, è costituita in parte da arte, in parte da scienza e in parte da insegnamento. È molto facile programmare in Visual Basic, ma solo fino a un certo punto. Per oltrepassare quel punto, che costituisce una specie di muro, bisogna conoscere i segreti e le tecniche arcane tramandate oralmente da programmatore a programmatore, bisogna sfogliare i forum online, assimilare documentazione proveniente dalle fonti più disparate e fare pratica. In questo libro sono condensati i miei anni di esperienza come sviluppatore, nonché i trucchi e i segreti che permettono di attraversare quel muro. Per la prima volta, in un unico luogo, sono raccolte tutte le informazioni di cui un programmatore in Visual Basic ha bisogno per creare sofisticare applicazioni professionali.

Che cosa serve

Per creare le applicazioni Visual Basic 6 esaminate in questo libro ovviamente bisogna avere installato una copia di Visual Basic 6. Nel Capitolo 1 viene fornito un esame dettagliato delle diverse versioni di Visual Basic e della relazione esistente tra Visual Basic e Visual Studio 98. In linea di massima parto dal presupposto che abbiate installato almeno la Professional Edition, però ci sono capitoli che hanno senso unicamente se si sta lavorando con PEnterprise Edition (tali capitoli vengono opportunamente evidenziati). Nonostante ciò, buona parte del materiale, sopratutto nelle prime parti del libro, è di aiuto anche a coloro che hanno installato la Learning Edition.

[1 libro è stato scritto per lettori di livello intermedio-avanzato. Nell'ambito di tale contesto ho fatto del mio meglio per renderlo utile al maggior numero possibile di lettori. Anche chi non ha mai programmato prima in Visual Basic può leggere questo libro, a patto che abbia già esperienze di programmazione in altri linguaggi. Coloro che sono già programmatori Visual Basic esperti troveranno numerosi segreti, suggerimenti, strumenti e tecniche.

Tuttii tipi di lettori trarranno beneficio dalle mie indicazioni su come creare codice di qualità, ma non raccomando questo libro come introduzione alla programmazione: chi non ha mai programmato prima è meglio faccia riferimento a un testo più semplice e riprenda in mano questo tra un po'!

Come usare questo libro

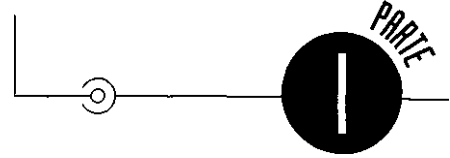
Se siete programmatori esperti e avete usato una precedente versione di Visual Basic, la strategia migliore probabilmente consiste nell'iniziare dal Capitolo 5 per poi proseguire in base agli argomenti di interesse.

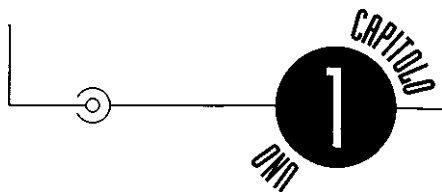
Coloro che cercano informazioni sulla programmazione in Windows facciano riferimento alla Parte II. Vale la pena anche di dare un'occhiata alla Parte VII per un'analisi dei migliori prodotti add-on per Visual Basic e di come è possibile usarli con Visual Basic 6 (probabilmente sarete al corrente del fatto che molto del successo di Visual Basic dipende dalla facilità con la quale terze parti possono fornire strumenti aggiuntivi di grande utilità).

Il mondo della programmazione di componenti è, per buona parte, il mondo di ActiveX e OLE. La programmazione di applicazioni ActiveX è facile ed eccitante in Visual Basic 6. Per informazioni su questo argomento fare riferimento alla Parte V.
Per conoscere la programmazione orientata agli oggetti (OOP) e per sapere come viene implementata in Visual Basic 6 bisogna fare riferimento al Capitolo 3, poi al Capitolo 4 e quindi procedere con la Parte III.
Combinando l'uso di OLE e i concetti di OOP si possono creare veri e propri controlli ActiveX mediante Visual Basic 6. Tali controlli ActiveX possono essere adoperati come componenti in altri ambienti di sviluppo come Visual C++, Delphi e FoxPro. Inoltre è facile aggiungerli alle applicazioni per il Web. Per informazioni sulla creazione di controlli ActiveX fare riferimento alla Parte VI.
Adesso più che mai è facile e divertente estendere l'ambiente Visual Basic. I documenti ActiveX sono forni Internet. Gli add-in o aggiunte sono componenti ActiveX che è possibile scrivere in Visual Basic e modificano l'IDE Visual Basic e il progetto Visual Basic corrente. I Wizard sono applicazioni che guidano un utente nello svolgimento di un'operazione. Nella Parte VII mostro come creare applicazioni Internet in Visual Basic, come creare documenti ActiveX che possano essere visti con un browser Internet e come creare propri Wizard e add-in Visual Basic.
Le nuove funzionalità per Internet vengono descritte nel Capitolo 5 e nel Capitolo 28.
I programmatori che non hanno mai sperimentato prima le gioie di Visual Basic è opportuno comincino dall'inizio: dopo aver letto questa introduzione, passino al Capitolo 1 e quindi al Capitolo 2. In tal modo potranno acquisire familiarità con la sintassi di Visual Basic e creare applicazioni importanti in men che non si dica.

Nota alla traduzione

Nella traduzione, si è scelto di attenersi alla versione inglese di Visual Basic 6: molti programmatori la usano anche in Italia ed è importante conoscerla se si vogliono sostenere gli esami di certificazione Microsoft. Per chi possiede la versione italiana, l'Appendice B fornisce due tabelle di corrispondenza fra le versioni inglese e italiana per tutte le espressioni utilizzate nel corso del libro (per le quali esiste una traduzione nell'ambiente VB: non tutto in effetti è tradotto).
Sul CD-ROM allegato al libro si troveranno poi i listati originali preparati dall'autore: nel corso del testo gli spezzoni di listato sono stati mantenuti inalterati, tranne per i commenti che sono stati tradotti.

PANORAMICA SU VISUAL BASIC

LA PIATTAFORMA

LA PIATTAFORMA

- Le varie edizioni di Visual Basic versione 6
- Visual Basic 6 e Visual Studio
- Installazione di Visual Basic versione 6
- Utilizzo della documentazione di VB6
- La Guida in stile HTML
- Windows e il Web
- Che cosa c'è di nuovo nella versione 6 di Visual Basic

Questo Capitolo descrive quello che è necessario sapere per iniziare ad usare Visual Basic. Incominceremo col dire quali sono le differenze fra le varie edizioni di Visual Basic. Poi, spiegheremo come Visual Basic si inserisce in Visual Studio, l'offerta primaria di Microsoft agli sviluppatori. Vedremo come installare Visual Basic 6 e usare la sua documentazione in linea. Infine, discuteremo alcune questioni generali come la Guida in stile HTML, e il rapporto tra Windows e il Web. Il capitolo termina con una panoramica delle novità di Visual Basic 6.

Le edizioni Learning, Professional e Enterprise di Visual Basic 6

Sono disponibili tre diverse edizioni di Visual Basic 6:

- Edizione Learning
- Edizione Professional
- Edizione Enterprise

Tutte e tre le edizioni sono applicazioni a 32 bit, nel senso che funzionano sotto Windows 98, Windows 95, o Windows NT, ma non sotto Windows 3.x. Producono solamente programmi a 32 bit da usarsi sotto questi sistemi operativi (e sul Web).

Visual Basic 4 è stata l'ultima versione di questo prodotto a supportare la creazione ài applicazioni a 16 bit (oltre a quelle a 32 bit).

L'edizione Learning è la meno costosa delle tre. Comprende i controlli di nucleo di Visual Basic, talvolta chiamati controlli intrinseci, il controllo *grid,* il controllo *outline, e* i controlli associati ai dati (ma non gli altri controlli compresi nell'edizione Professional). All'edizione Learning mancano alcune funzionalità necessarie alla creazione di applicazioni professionali.

L'edizione Professional comprende tutto ciò che fa parte dell'edizione Learning. Viene però fornita con un'ampia collezione di controlli aggiuntivi, tra i quali vi sono i controlli 3D, un controllo di pulsante animato, un controllo di comunicazioni, un controllo ListView, un controllo MAPI, un controllo ProgressBar, un controllo Toolbar, e molti altri, compresi tutti i controlli Internet. Inoltre, viene fornita con Crystal Report Writer. L'edizione Professional ha l'importante capacità, assente nell'edizione Learning, di creare applicazioni con componenti ActiveX e controlli ActiveX.

L'edizione Enterprise è rivolta ad aiutare gruppi di programmatori professionisti a produrre applicazioni robuste, eventualmente client/server, in un ambiente aziendale.

L'edizione Enterprise comprende tutto ciò che fa parte dell'edizione Professional più un *repository* (un deposito) di oggetti, che serve a organizzare i componenti in un ambiente aziendale, l'Automation Manager, il Component Manager, Visual SourceSafe (uno strumento di gestione versioni), strumenti di accesso e di gestione di database, strumenti di sviluppo client/server e altro.

Una caratteristica particolarmente importante dell'edizione Enterprise è che abilita a creare server OLE di automazione remota che possono venire eseguiti a distanza tramite una rete.

Per mantenere le cose semplici, questo libro suppone che il lettore stia lavorando almeno con l'edizione Professional. Come strumento di sviluppo a livello avanzato, l'edizione Learning non è proprio sufficiente. Molte delle caratteristiche più interessanti di Visual Basic (per esempio la possibilità di creare applicazioni basate su Dynamic HTML mediante Visual Basic) non sono affatto disponibili con l'edizione Learning.

Verranno descritte anche alcune delle caratteristiche specifiche dell'edizione Enterprise. Per esempio, il Capitolo 12 discute gli strumenti di gestione delle versioni che sono forniti solamente con l'edizione Enterprise. Quando accadrà, verrà indicato chiaramente che la discussione si rivolge solamente a chi sta lavorando con l'edizione Enterprise.

Visual Basic 6 e Visual Studio

Con la versione 6 di Visual Basic, Microsoft ha integrato il prodotto VB nella sua serie di strumenti per lo sviluppo di applicazioni Visual Studio. Essenzialmente, VB6 è un'applicazione plug-in di Visual Studio 6, e come tale condivide documentazione, librerie, strumenti, e, fino a un certo punto, interfacce utente con le altre applicazioni che fanno parte di Visual Studio. La maggior parte degli sviluppatori useranno Visual Basic 6 congiuntamente a Visual Studio 6.

Oltre a Visual Basic, le applicazioni Visual Studio comprendono i seguenti componenti:

- Visual C++ 6.0
- Visual InterDev 6.0
- Visual J++ 6.0
- Visual FoxPro 6.0

Fra l'altro, si vedrà che nella documentazione talvolta si fa riferimento ai prodotti della versione 6 di Visual Studio con il suffisso 98 (per esempio, Visual Studio 98 e Visual Basic 98) ma questo non deve trarre in inganno. Si tratta in realtà degli stessi prodotti.

Proprio come Visual Basic, anche Visual Studio esiste in edizione Professional e in edizione Enterprise. Entrambe le versioni comprendono degli strumenti di sviluppo che verranno spiegati nel Capitolo 11. Essenzialmente, gli strumenti comuni forniti con l'edizione Professional di Visual Studio sono gli stessi (o versioni aggiornate) di applicazioni che in passato venivano fornite come SDK per Win32. Gli strumenti dell'edizione Enterprise di Visual Studio sono rivolti ad aiutare lo sviluppo di applicazioni client/server multi-tier (cioè a più livelli).

L'installazione di Visual Basic 6

Per eseguire l'edizione Professional di Visual Basic 6, c'è bisogno, ovviamente, di un sistema su cui funzioni Windows 98, 95, Windows 2000, o NT 4.0.
Microsoft suggerisce un minimo di 16 MB di RAM. Come consueto con questo genere di raccomandazioni, è meglio avere più RAM: possibilmente almeno 32 MB.
Un'installazione completa dei file comuni di Visual Studio e dell'edizione Enterprise di VB6 richiede circa 130 MB di spazio sul disco fisso; ci sarà bisogno di almeno 50 MB liberi sull'unità di avvio per installare con successo il prodotto. L'installazione completa del prodotto Visual Studio richiede molto più spazio.

Come regola generale, è una buona idea collaudare la propria applicazione su un'ampia gamma di hardware, di varie velocità. I programmi che funzionano accettabilmente sulla propria macchina potrebbero essere intollerabilmente lenti su sistemi meno potenti. In generale, gli sviluppatori tendono ad avere hardware veloce e moderno. Questo potrebbe non essere affatto il caso per gli utenti delle applicazioni.

Le edizioni Professional o Enterprise di VB6 verranno solitamente installate come parte dell'installazione di Visual Studio.
Il programma di setup, quando parte sul primo dei due CD-ROM di Visual Studio, verifica l'esistenza nel sistema di una versione aggiornata di Microsoft Internet Explorer (4.01 o successiva). Questo programma è necessario per l'installazione di Visual Studio. Se non è presente nel sistema, il setup wizard di Visual Studio lo installa e, dopo un riavvio del sistema, riprende l'installazione di Visual Studio.

Si possono selezionare i singoli prodotti che si desidera installare, come mostrato nella Figura 1.1, o esercitare un controllo ancora più fine su ciò che sarà effettivamente installato usando l'opzione di setup personalizzato, come mostrato nella Figura 1.2.

Figura 1.1 *Visual Studio permette di selezionare i singoli prodotti da installare.*

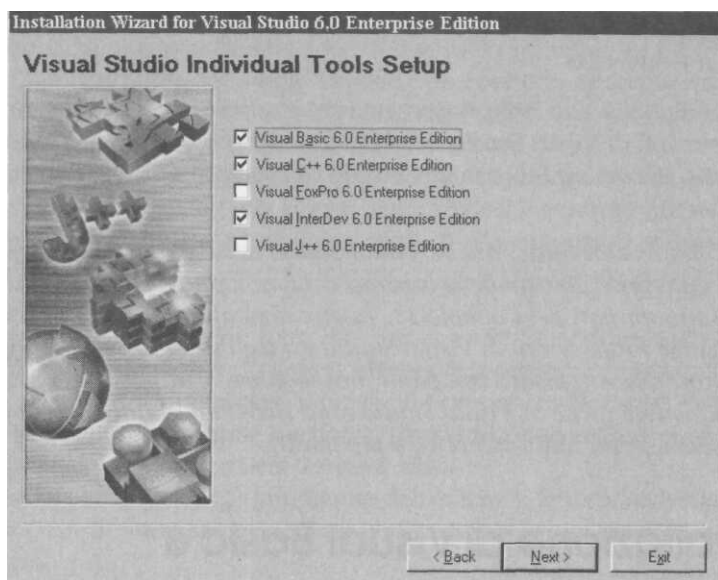



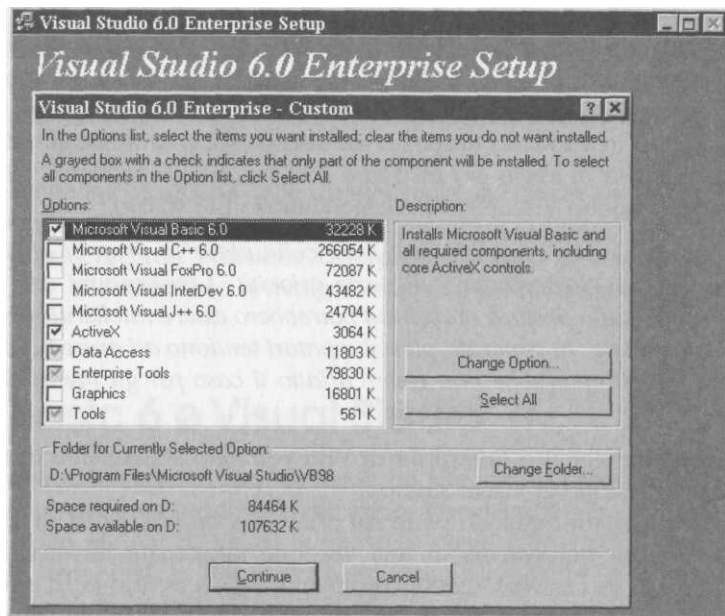
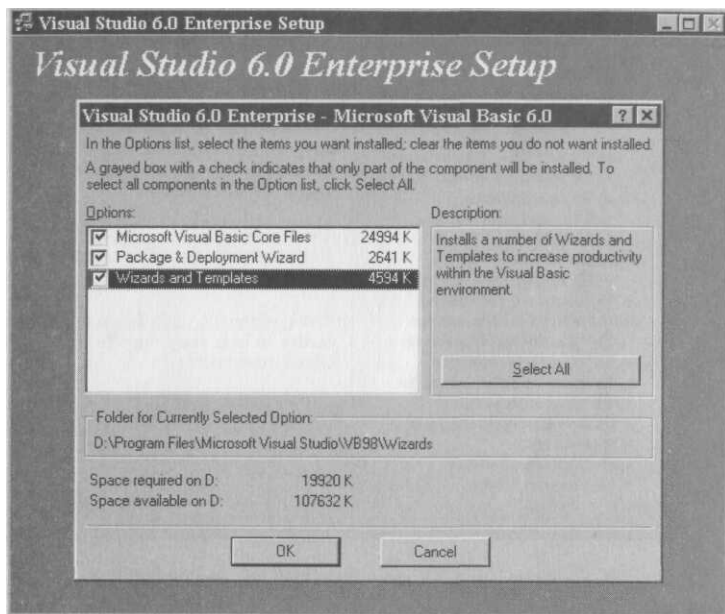
Figura 1.2 *Selezionando le scelte di installazione personalizzata (Custom) si ha un controllo molto accurato su ciò che Visual Studio installa effettivamente.*

Se è stata selezionata l'installazione personalizzata, si possono effettuare ulteriori scelte su quali componenti di Visual Basic si vorrebbe che fossero installati, come si può vedere nella Figura 1.3.

Figura 1.3 *La finestra di dialogo relativa atte opzioni di installazione personalizzata di Visual Basic 6 serve per selezionare i componenti da installare.*

L'installazione di Visual Basic 6

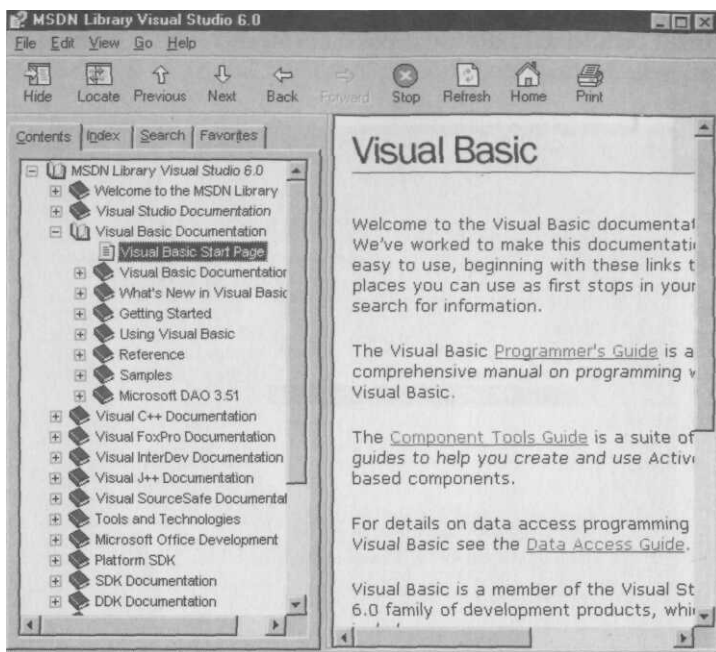
La documentazione di Visual Basic 6 viene fornita da un'edizione speciale della libreria Microsoft Developers Network (MSDN), che viene visualizzata con un'interfaccia utente simile a Explorer (Figura 1.4).

L'edizione per Visual Studio di MSDN deve venire installata mediante il suo programma di setup dopo aver installato Visual Studio.

La Guida in stile HTML

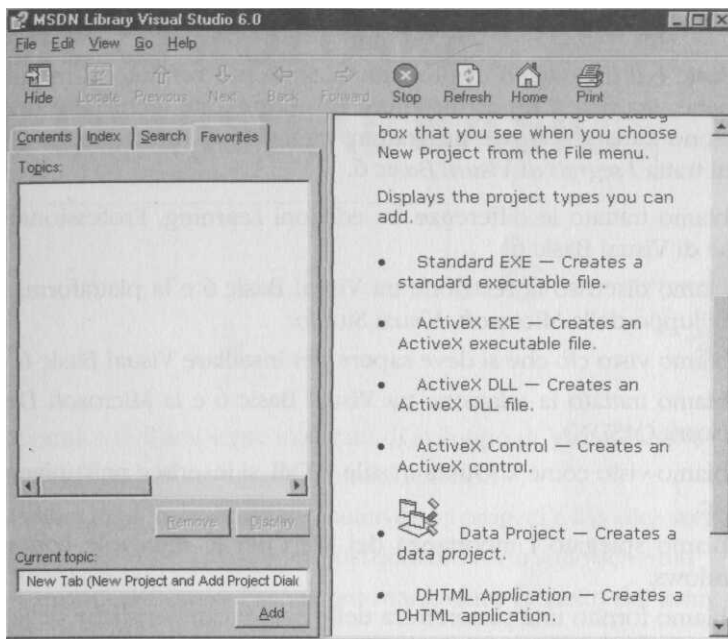
La documentazione che si trova nell'edizione per Visual Studio di MSDN, che appare quando si seleziona la Guida in linea ovunque all'interno dell'ambiente di sviluppo Visual Basic, viene presentata nel modernissimo stile HTML invece che nel vecchio stile dei file della Guida di Windows.

Questa è una situazione in cui, come si suol dire, "i vantaggi dell'uno non sono gli svantaggi dell'altro". I file della Guida di Windows hanno alcune caratteristiche che mancano nella Guida HTML. D'altro canto, migrare la documentazione dal formato di file della Guida di Windows al formato HTML promuove gli standard aperti che possono venire usati per tutto il Web. Inoltre, i file della Guida di Windows richie-

Figura 1.4
La documentazione di VB6 è disponibile in un'edizione pedale di MSDN.

devano notoriamente molto tempo per crearli e compilarli. Per ulteriori informazioni sulla creazione di file di Guida HTML, vedere il Capitolo 35.
Sebbene la Guida HTML non abbia tutti i sussidi di interfaccia e di navigazione disponibili nella Guida di Windows, questo fatto può essere compensato usando gli strumenti disponibili nel visualizzatore in stile Explorer di MSDN, come mostrato in Figura 1.5.

Windows e il Web

Come vola il tempo! Sembra ieri quando sono stati rilasciati Windows 95 e NT 4.0 e la massa degli utenti di Windows è stata introdotta ai sistemi operativi a 32 bit e alle loro nuove interfacce utente. Visual Basic 6 è la seconda release primaria di VB pienamente a 32 bit. Oggi, Windows a 32 bit è maturo e, oseremmo dire, un po' di mezza età.
Così VB6, mentre rappresenta un solido ed efficiente strumento per lo sviluppo standard per Windows, guarda avanti in vari modi verso ciò che potrebbe diventare il "sistema operativo" del futuro: il Web. Ciò viene effettuato mediante numerosi nuovi strumenti e caratteristiche, ma soprattutto traendo vantaggio da "ganci" costruiti in Internet Explorer. Questo abilita gli sviluppatori VB a fare cose interessanti, come "far persistere" dei dati quando un utente naviga fuori da una pagina HTML, e creare applicazioni DHTML con VB.

Figura 1.5
Sipuò navigare facilmente la documentazione della Guida HTML fornita con VB6 usando il visualizzatore MSDN.

Questo sviluppo in VBó va di pari passo con le tendenze di Windows stesso. La caratteristica Active Desktop che si può attivare in Windows 95 con Explorer 4, e che fa parte di Windows 98, rappresenta essenzialmente la "Webizzazione" dell'interfaccia utente di Windows.

Panoramica delle nuove caratteristiche di Visual Basic 6

Tra le nuove caratteristiche chiave di VB6 ci sono:

- *Strumenti migliorati per l'accesso ai dati e per applicazioni client/server*
- *Nuovi strumenti e nuove caratteristiche per Internet e il Web*
- *Nuovi controlli e controlli migliorati*
- *Strumenti potenziati per la creazione di componenti*
- *Nuove caratteristiche linguistiche*
- *Nuovi wizard e wizard potenziati*

Si possono trovare informazioni dettagliate su molte caratteristiche nuove di VBó nel Capitolo 5. Il lavoro con Visual Basic 6 e il Web è trattato nel Capitolo 28.

Riepilogo

Visual Basic è il linguaggio di programmazione più venduto al mondo. I manuali vanno bene, ma fino a un certo punto. Esistono molte tecniche e trucchi nascosti che possono aiutare a creare programmi migliori più rapidamente. Questo è tutto ciò di cui tratta *I segreti di Visual Basic 6.*

- Abbiamo trattato le differenze fra edizioni Learning, Professional e Enterprise di Visual Basic 6.
- Abbiamo discusso la relazione tra Visual Basic 6 e la piattaforma primaria di sviluppo della Microsoft, Visual Studio.
- Abbiamo visto ciò che si deve sapere per installare Visual Basic 6.
- Abbiamo trattato la relazione tra Visual Basic 6 e la Microsoft Developers Network (MSDN).
- Abbiamo visto come la Guida in stile HTML si inserisce nell'universo Visual Basic.
- Abbiamo spiegato l'importanza del Web per la mutevole concezione di Windows.
- Abbiamo fornito una panoramica delle nuove caratteristiche della versione 6 di Visual Basic.

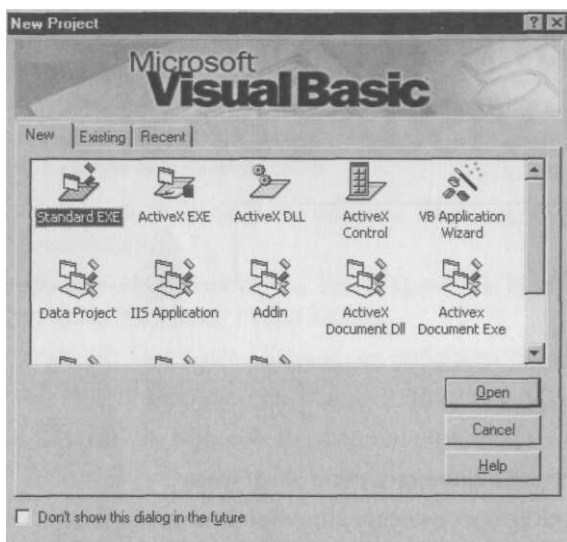
COME SFRUTTARE L'IDE DI VISUAL BASIC

- Una panoramica dell'ambiente integrato di sviluppo di Visual Basic 6
- Uso della Toolbox per aggiungere controlli ai form
- Uso del Project Explorer per navigare attraverso i progetti e il codice sorgente
- Uso della finestra Form Layout per posizionare i form sullo schermo
- Uso dei comandi del menu Format per manipolare l'aspetto dei form, dei controlli, e delle pagine di proprietà
- Uso efficace della finestra di codice
- Spostamento rapido nel codice sorgente con l'Object Browser
- Uso del Menu Editor per aggiungere menu ad un form
- Uso degli strumenti di debug di Visual Basic
- Creazione di file eseguibili compilati

Visual Basic ha fatto una lunga strada da quei primi giorni quando era chiamato in codice "Thunderbolt" e funzionava sotto Windows 3.0. Questo capitolo inizia con una panoramica dell'ambiente integrato di sviluppo di Visual Basic (in inglese, Integrated Development Environment, abbreviato in IDE). Per chi non fosse ferrato nell'uso di Visual Basic (o stesse iniziando ad usare la versione 6) questa panoramica fornisce un'idea di come effettuare le operazioni di base prima di inoltrarsi in argomenti più avanzati. Il capitolo è, grosso modo, organizzato nello stesso ordine in cui si intraprenderebbe una sessione di programmazione: dapprima si apre il progetto (oppure si inizia un nuovo progetto), poi si usano gli strumenti appositamente forniti nell'IDE per lavorarci, e, infine, si compila il programma.

Panoramica dell'IDE di Visual Basic

L'ambiente di sviluppo di Visual Basic è piuttosto immediato e facile da usare dopo averne fatto conoscenza. Quando si lancia Visual Basic, di default si vede la finestra di dialogo *New Project* mostrata in Figura 2.1. Si può usare questa finestra di dialogo per aprire un progetto esistente oppure un nuovo progetto. Una terza scheda facilita l'apertura di progetti su cui si è lavorato di recente.

Figura 2.1
La prima volta che si avvia Visual Basic 6 si vede la finestra di dialogo New Project.

Quando si sceglie *New* dal menu *File* di Visual Basic, la finestra di dialogo *New Project* che compare consiste della sola scheda *New;* non comprende le schede *Existing* e *Recent* che si vedono nella Figura 2.1.

Se si imposta l'opzione Don't Show this Dialog in the Future *sulla finestra di dialogo iniziale* New Project, *non si vedrà questa finestra di dialogo le volte successive che si avvia Visual Basic. In tal caso, di default VB crea automaticamente un progetto di Standard EXE quando si avvia. Per ritornare alla finestra di dialogo iniziale* New Project, *bisogna scegliere* Options *dal menu* Tools, *quindi scegliere la scheda* Environment, *e infine selezionare l'opzione* Prompt for Project *nel gruppo* When Visual Basic Starts.

La finestra di dialogo New Project

Per impostazione predefinita, la finestra di dialogo *New Projects* dell'edizione Enterprise presenta 11 opzioni (non tutte saranno presenti nelle edizioni Professional e Learning).

I progetti che si aggiungono alla directory Templates\Project di Visual Basic appaiono come opzioni di template (modelli) nella finestra di dialogo New Project. *(In altre parole, tra le opzioni che si vedono sulla scheda* New *della finestra di dialogo* New Project *mostrata nella Figura 2.1 ci sono sia i modelli sia i diversi tipi di progetti). Nel Capitolo 5 vedremo come aggiungere i proprì progetti modello alle scelte disponibili.*

Le seguenti scelte di progetto sono disponibili subito dopo l'installazione di VB6:

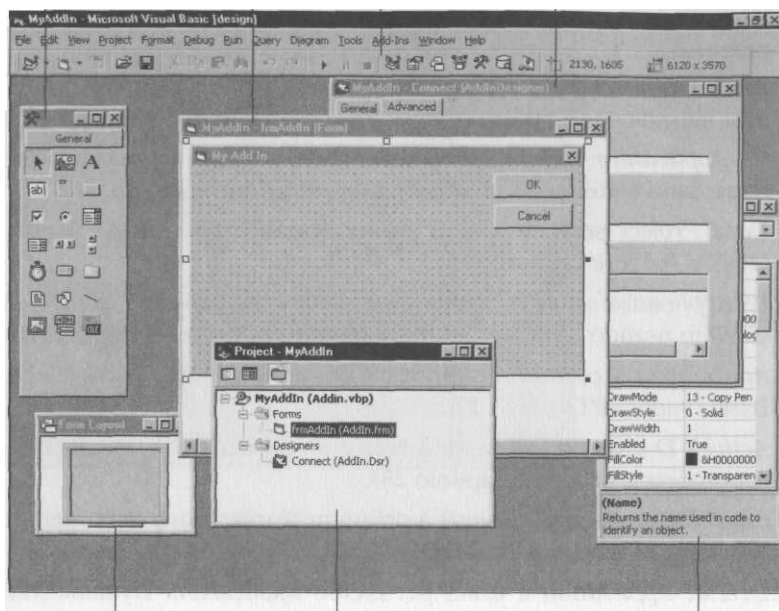
- *Standard EXE* serve a creare un progetto eseguibile standard.

- *ActiveX EXE* serve a creare un'applicazione server OLE "out-of-process" (fuori dal processo).
- *ActiveX DLL* serve a creare un'applicazione server OLE "in-process" (nel processo).
- *ActiveX Control* serve a creare un controllo ActiveX (un file .Ocx). Per ulteriori informazioni, vedere la Parte VI, "Creazione di controlli ActiveX".
- *VB Application Wizard* guida attraverso le scelte di progettazione iniziali riguardanti l'interfaccia utente di un'applicazione standard.
- *Data Project* serve a creare un'applicazione che funzioni con i Data Objects (vedere il Capitolo 5).
- *IIS Application* serve a creare applicazioni che possano essere pubblicate sul Web usando l'Internet Information Server (vedere il Capitolo 28).
- *Addin* aiuta a costruire le proprie aggiunte (in inglese, add-in) per Visual Basic (vedere il Capitolo 29).
- *ActiveX Document DLL* serve a creare un'applicazione documento ActiveX "in-process" (vedere il Capitolo 28).
- *ActiveX Document EXE serve* a creare un'applicazione documento ActiveX "out-of-process" (vedere il Capitolo 28).
- *DHTML Application è* usato per creare applicazioni Dynamic HTML che possano venire eseguite da Internet Explorer (vedere il Capitolo 28).

Le applicazioni server OLE "in-process" vengono eseguite come pane dello stesso thread dell'applicazione client che le ha invocate, mentre i server OLE "out-of-process" richiedono il loro thread di esecuzione autonomo. Per ulteriori informazioni, vedere la Parte V, "Utilizzo di ActiveX".

Gli elementi dell'IDE

Dopo aver aperto un progetto in VB6, si vedrà qualcosa di simile a ciò che è mostrato nella Figura 2.2. (Si noti che si può configurare l'aspetto della maggior parte degli elementi dell'IDE; l'aspetto effettivo della schermata dipende dalle impostazioni che si sono selezionate e da come si sono disposte le cose.)
In cima allo schermo c'è la *barra del titolo* (chiamata anche *caption bar),* che visualizza il nome del progetto aperto e indica se si sta lavorando in modalità progettazione, modalità esecuzione, o modalità interruzione.
Sotto la barra del titolo c'è la barra dei menu, che dà accesso ai comandi con cui si possono costruire i propri progetti. La barra dei menu di Visual Basic contiene i menu *File, Edit, View, Project, Format, Debug, Run, Tools, Add-Ins, Window e Help* (nella versione italiana; nell'ordine, sono: *File, Modifica, Visualizza, Progetto, Formato, Debug, Esegui, Strumenti, Aggiunte, Finestra* e *Guida).*
Sotto la barra dei menu c'è la *toolbar (barra degli strumenti).* La toolbar contiene i pulsanti che permettono di eseguire rapidamente delle comuni operazioni di programmazione, tra cui aggiungere un nuovo progetto; aggiungere un nuovo modulo; aprire l'editor dei menu; aprire un progetto esistente; salvare il progetto corrente;

Figura 2.2 *Si può vedere che l'IDE di VB6 è un concentrato dipotenza.*

tagliare, copiare, incollare, trovare, annullare, e rifare; avviare, sospendere, o terminare l'esecuzione di un'applicazione; mostrare il Project Explorer; mostrare la finestra *Properties;* mostrare la finestra *Form Layout;* mostrare PObject Browser; e mostrare la Toolbox.

Alcuni dei pulsanti della toolbar adesso richiamano delle caselle di riepilogo di opzioni. Per esempio, quando si fa clic sul pulsante *Add Module,* compare una casella di riepilogo a discesa, dalla quale si può scegliere che tipo di modulo si vorrebbe aggiungere.

Altri elementi dell'IDE mostrati in Figura 2.2 (e che compaiono di default quando si apre un nuovo progetto Standard EXE) sono:

- La Toolbox (casella degli strumenti), che visualizza i controlli ActiveX attualmente disponibili (e altri oggetti che possono essere inseriti nei propri form VB)
- La finestra *Immediate,* che viene usata in fase di debug per mostrare le informazioni prodotte dalle istruzioni di debug inserite nel codice, o richieste digitando comandi interattivamente nella finestra
- II Project Explorer, che serve a navigare fra i moduli di un progetto (e fra i progetti di un gruppo di progetti)
- La finestra *Form Layout,* che serve a posizionare i form sullo schermo

La finestra *Properties,* che serve a impostare le proprietà dei form e dei controlli in fase di progettazione

Un designer di form, che serve a manipolare l'aspetto di un form

risono designer (=progettisti) per manipolare l'aspetto di altri tipi di moduli oltre ai form Per esempio, si può usare un designer di controlli d'utente per creare l'aspetto in fase di esecuzione di un controllo ActiveX. Le applicazioni basate su controlli ActiveX vengono costruite intorno ai moduli controlli d'utente nello stesso modo in cui le applicazioni eseguibili standard vengono costruite intorno ai moduli form. Non si troveranno differenze sostanziali fra i vari tipi di designer, eccetto che operano su diversi tipi di moduli.

Nelle versioni di Visual Basic precedenti a VB5, le funzionalità del designer di form e della finestra *Form Layout* erano espletate da una sola finestra, che era lo spazio di lavoro primario di VB. Con VB6, l'uso dei designer è diventato abituale. Sebbene i designer possano avere un aspetto diverso sullo schermo, ci si dovrebbe attendere un designer come interfaccia utente primaria per la manipolazione dei più disparati tipi di oggetti.

Inizio di un nuovo progetto o apertura di un progetto esistente

Come detto sopra, quando si avvia Visual Basic, di default compare una finestra di dialogo che permette di aprire un progetto nuovo o uno esistente. Se l'opzione di ambiente che visualizza questa finestra di dialogo all'avvio è stata disattivata, o se VB è già aperto, è pur sempre facile iniziare un nuovo progetto in VB: basta scegliere *New Project* dal menu *File.*
Per aprire un progetto esistente, bisogna scegliere *Open Project* dal menu *File* (oppure premere da tastièra la combinazione Ctrl+O), usare la finestra di dialogo comune che compare per spostarsi nella directory dove vive il progetto, sceglierlo, e poi fare clic sul pulsante *Open.*
Per comodità, la finestra di dialogo *Open Project* adesso fornisce una scheda *Recent,* da cui si può aprire qualunque progetto su cui si sia lavorato di recente, come si vede nella Figura 2.3.

Salvataggio dei progetti

Quando si salva un progetto Visual Basic, o un gruppo di progetti, l'ambiente VB fa passare attraverso una serie di finestre di dialogo comuni *Save* (a seconda dei tipi di noduli contenuti nel progetto). Per salvare un progetto e i suoi file associati, scegliere *Save Project* (o *Save Project As)* dal menu *File,* oppure fare clic sul pulsante *Save* della barra degli strumenti.

Se si sta salvando un gruppo di progetti (che è semplicemente un insieme di più progetti aperti nello spazio di lavoro) le voci sul menu *File* si modificano rispettivamente in *Save Project Group* e *Save Project Group As.*

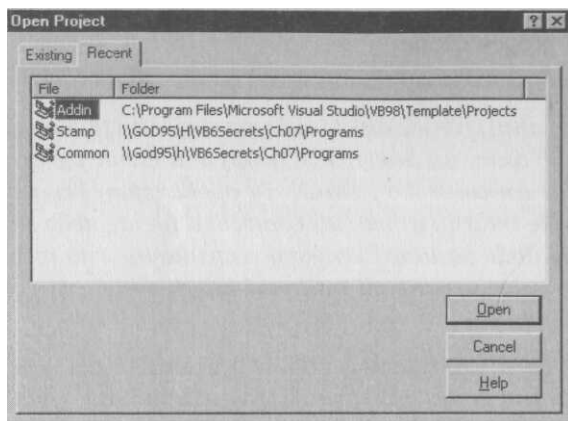
Figura 2.3
Si può usare la scheda Recent della finestra di dialogo Open Project per aprire qualunque progetto su cui si sia lavorato di recente.

Le finestre di dialogo fanno passare attraverso i contenuti del proprio intero progetto o gruppo di progetti, dapprima permettendo di salvare i file dei form (.Frm), poi i file dei moduli (.Bas), i moduli delle classi (.Cls), i documenti ActiveX (.Dob), i controlli d'utente (.Ctl), le pagine di proprietà (.Pag), e infine lo stesso (o gli stessi) file di progetto (.Vbp) e, se ne esiste uno, il gruppo di progetto (.Vbg). Più file di tipi diversi possono avere lo stesso nome (purché abbiano una diversa estensione) se lo si gradisce.
La Tabella 2.1 elenca tutti i tipi di file (con le rispettive estensioni) che possono far parte di un progetto o gruppo di progetti Visual Basic.

Tabella 2.1 *Tipi di file che costituiscono un progetto Visual Basic.*

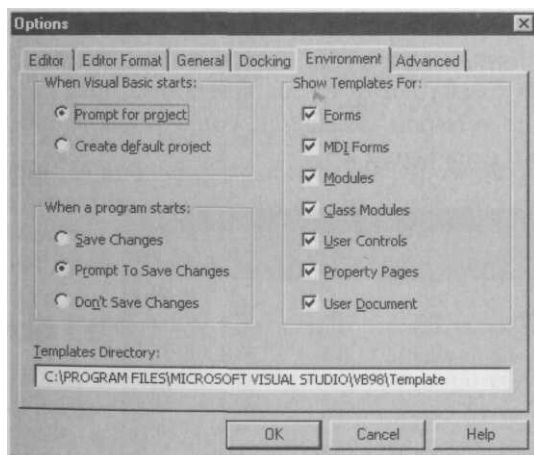
Estensione	Scopo
.Bas	Modulo sorgente di codice
.Cls	Modulo sorgente di classe
.Ctl	File di controllo d'utente
.Ctx	File binario di controllo d'utente
.Dca	Cache di designer attiva
.Dep	File di dipendenze del Setup Wizard
.Dob	File di form di documento d'utente
.Dox	File di form binario di documento d'utente
.Dsr	File di designer attivo
.Dsx	File binario di designer attivo
.Frm	File di form
.Frx	File di form binario
.Log	File di log per errori di caricamento
.Oca	Cache di Control Typelib
.Pag	File di pagina di proprietà
.Pgx	File di pagina di proprietà binario
.Res	File di risorse
.Swt	File di modelli del Setup Wizard di Visual Basic

Estensione	Scopo
.Tlb	File di Remote automation Typelib
.Vbg	Gruppo di progetti di Visual Basic
.Vbl	File di licenza per controllo d'utente
.Vbp	Progetto di Visual Basic
.Vbr	File di registrazione di Remote automation
.Vbw	Spazio di lavoro di progetto Visual Basic
.Vbz	File di lancio di Wizard

A qualcuno potrebbe interessare sapere che i gruppi di progetti di Visual Basic (i file Vbg) non possono essere annidati. In altre parole, un gruppo di progetti può contenere numerosi progetti, ognuno dei quali contiene numerosi moduli, ma non può contenere un altro gruppo di progetti.

Impostazione delle opzioni di ambiente, di editor e generali

L'IDE di Visual Basic può venire personalizzato in molti modi. Per farlo, iniziare scegliendo *Options* dal menu *Tools.* Comparirà una finestra di dialogo a più schede, come mostrato nella Figura 2.4. Usando le schede di questa finestra di dialogo, si possono impostare le opzioni come descritto in ogni scheda.

Figura 2.4
Ecco la finestra di dialogo Options con la scheda Environment in vista.

La scheda Environment

La scheda *Environment* serve a specificare gli attributi del proprio ambiente di sviluppo Visual Basic. Le modifiche apportate qui vengono salvate nel file del

Se è selezionato *Promptfor Project,* ogni volta che si avvia Visual Basic viene proposta la scelta del tipo di progetto che si vuole aprire (vedere "Inizio di un nuovo progetto o apertura di un progetto esistente" più sopra in questo capitolo).
Se è selezionato *Create Default Project,* ogni volta che si avvia Visual Basic viene aperto un progetto di eseguibile di default (cioè uno standard EXE).
Se è selezionato *Save Changes* sotto *When a Program Starts,* l'esecuzione di un progetto salva automaticamente ogni modifica apportatagli dall'ultimo salvataggio. Se si sta eseguendo un nuovo file che non è mai stato salvato, comparirà la finestra di dialogo comune *Save As,* in modo da poter dare un nome e una posizione al progetto.
Se è selezionato *Prompt To Save Changes,* l'esecuzione di un progetto visualizza una finestra di dialogo che chiede se si vuole salvare il materiale non salvato. Se si seleziona *Yes,* si può salvare il progetto e i suoi file. Se si seleziona *No,* Visual Basic esegue il progetto usando l'immagine in memoria, ma non salva nessuna modifica.
Se è selezionato *Don't Save Changes,* quando si esegue il progetto, Visual Basic esegue l'immagine in memoria del progetto e non salva le modifiche.

La migliore impostazioneper questa opzione è Prompt To Save Changes. *Ilpericolo di eseguire progetti non salvati è che si possa perdere pane del proprio lavoro. Questa impostazione ricorda di salvare il lavoroprima di eseguire ilprogetto, senza costringere afarlo in situazioni in cui sarebbe solamente una seccatura.*

Le caselle di controllo *Show Templates For* (vedere il lato destro di Figura 2.4) permettono di determinare se si vuole che i template per i moduli siano disponibili quando si aggiungono moduli ad un progetto.
Vediamo come funzionano, prendendo un modulo di form come esempio. Per aggiungere un modulo di form, scegliete *Add Form* dal menu *Project.* Se nel riquadro *Show Templates For* la casella *Forms* è stata selezionata, tutti i file .Frm della directory \Template\Forms verranno visualizzati come possibili modelli per un nuovo form, come mostrato nella Figura 2.5.

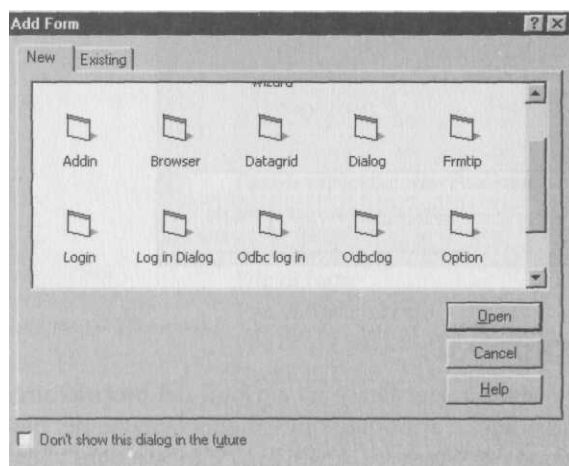
Figura 2.5
Sipossono usare i template per aggiungere nuovi moduli che comprendano molte caratteristiche fin dall'inizio.

Ovviamente, essere in grado di usare i modelli forniti da VB è una grande comodità. Ma si può fare un altro passo avanti e aggiungere i modelli personalizzati; per esempio, tutti i moduli di classe potrebbero includere del codice standard per le proprietàche si è soliti usare e tutti i moduli di forni potrebbero essere basati su particolari scelte di colori.
Tra l'altro, si può usare la scheda *Environment* della finestra di dialogo *Options* anche per modificare la posizione della struttura di default dei modelli. (La directory di default dei modelli è installata sotto la directory in cui è stato installato VB6 che, di default, è \Program Files\Microsoft Visual Studio\VB98.)
La Tabella 2.2 mostra i tipi di moduli che possono essere basati su dei template e la posizione di default dei modelli relativi.

Tabella 2.2 *Moduli dei relativi template.*

Tipo di modulo	Posizione di default dei template
Form	Template \Forms
Form MDI	TemplateMDIForms
Moduli di codice (file .Bas)	Template\Code
Moduli di classe	Template\Classes
Menu	Template\Menus
Progetti	Template\Projects
Controlli d'utente	Template\UserCtls
Pagine di proprietà	Template\Proppage
Documenti d'utente	Template\UserDocs

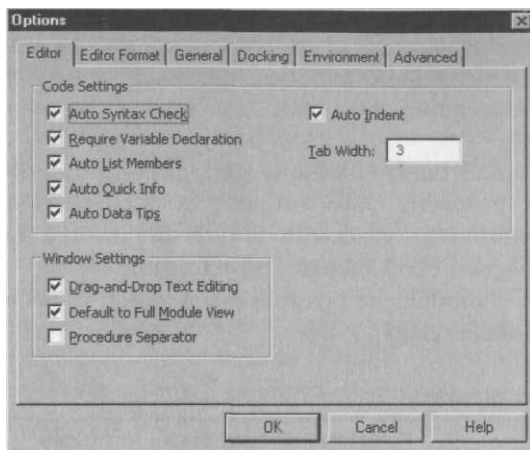
È possibile che non tutte queste sottodirectory di template siano state create quando è stato installato VB6, nel qual caso si dovrà crearle se si vogliono aggiungere moduli di modello personalizzati.
Oltre ai moduli, in una directory di template si possono porre dei wizard (autocomposizioni o procedure guidate), i file .Vbz. I wizard compariranno nella finestra di dialogo *AddForm* mostrata nella Figura 2.5. Se l'utente ne seleziona uno, questo si attiva e aiuta l'utente a creare uno specifico tipo di modulo.

La scheda Editar

La scheda *Editar,* mostrata nella Figura 2.6, serve a configurare il comportamento dell'editor di codice.

Il riquadro *Code Settings* abilita a controllare importanti aspetti della funzionalità dell'editor di codice:

- L'opzione *Auto Syntax Check* imposta la verifica automatica della sintassi dopo il completamento di ogni riga di codice.
- L' opzione *Require Variable Declaration* aggiunge l'istruzione Option Explicit all'inizio di ogni modulo. Questa istruzione impone che tutte le variabili usate siano dichiarate esplicitamente. È buona pratica di codifica abilitare questa opzione così da pretendere la dichiarazione delle variabili.

Figura 2.6
La scheda Editar serve a modificare il comportamento della finestra dell'editar di codice.

- Quando le opzioni *Auto List Members, Auto Quick Info* e *Auto Data Tips* sono abilitate, si riceveranno interattivamente informazioni sintattiche man mano che si inserisce il codice.

Inoltre, si può usare la scheda *Editor* per abilitare la modifica di testo tramite trascinamento (in inglese, drag-and-drop); impostare il numero di spazi che corrispondono a una tabulazione; impostare il rientro automatico; impostare la visualizzazione del codice per modulo intero o al contrario procedura per procedura; e, se si sta usando la visualizzazione per modulo intero, aggiungere o togliere i separatori di procedura.

La scheda Editor Format

La scheda *Editor Format,* mostrata nella Figura 2.7, serve a configurare l'aspetto dell'editor di codice.
Questa è la sede in cui si impostano il tipo e la dimensione dei caratteri per la finestra di codice, e in cui si possono specificare i colori del codice per diversi generi di testo, per esempio *Syntax Error Text, Keyword Text* e così via.

La scheda General

La scheda *General,* mostrata nella Figura 2.8, serve a specificare le impostazioni di griglia, le impostazioni di gestione degli errori e a compilare le impostazioni per il progetto Visual Basic corrente.
Se è impostata l'opzione *Show ToolTips,* compariranno degli aiuti a fumetto per le voci della barra degli strumenti e della Toolbox. Se è impostata l'opzione *Collapse Proj. Hides Windows,* le finestre del modulo vengono nascoste quando un progetto viene ridotto a icona nel Project Explorer.
Le impostazioni *Error Trapping* controllano come vengono gestiti gli errori nell'ambiente di sviluppo Visual Basic. Queste impostazioni non vengono salvate per ogni progetto, perciò impostare questa opzione influenza tutti gli esemplari di

Figura 2.7
la scheda Editar Format serve a determinare come viene visualizzato il codice.

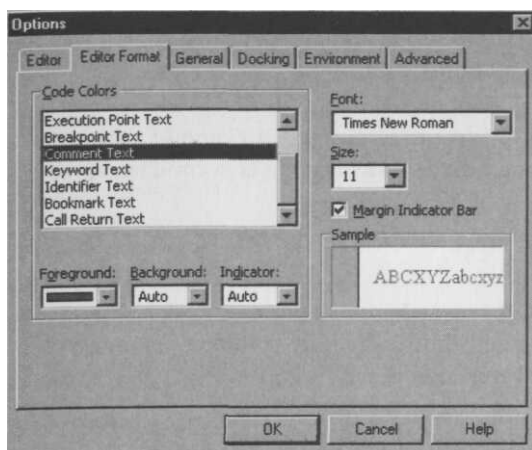
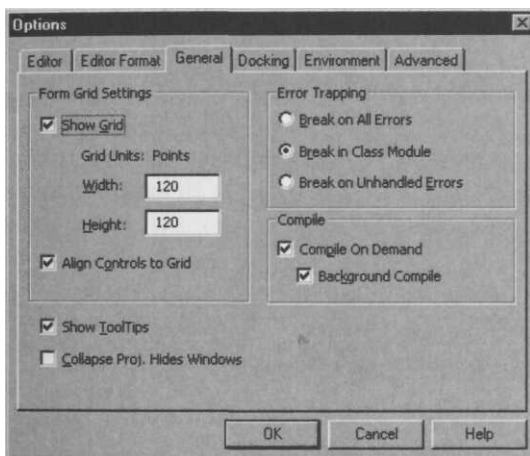



Figura 2.8
La scheda General serve a specificare il comportamento della griglia e a compilare le impostazioni per il progetto VB corrente.

Visual Basic avviati dopo aver modificato l'impostazione. Si può scegliere fra tre possibili impostazioni di cattura degli errori:

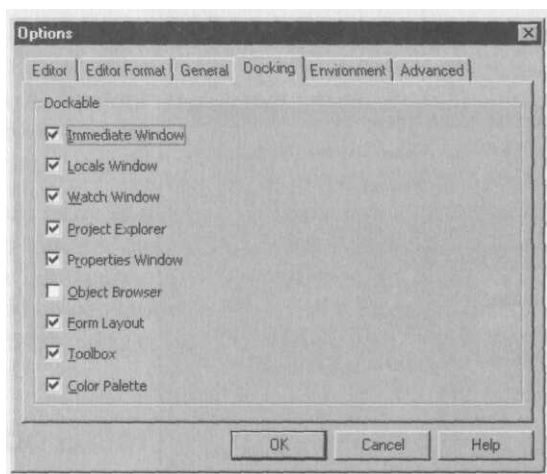
- Se è impostato *Break on All Errors,* ogni errore fa entrare il progetto in modalità interruzione.
- Se è impostato *Break in Class Module,* ogni errore non gestito prodotto in un modulo di classe fa entrare il progetto in modalità interruzione alla riga di codice del modulo di classe che ha prodotto l'errore.
- Se è impostato *Break on Unhandled Errors,* ed è attivo un gestore di errore, l'errore viene catturato senza entrare in modalità interruzione. Se non ci sono gestori di errore attivi, l'errore fa entrare il progetto in modalità interruzione.

Per ulteriori informazioni sulla gestione di errori, si veda il Capitolo 15, "Gestione degli errori".

Se si sta effettuando il debug di un progetto server ActiveX facendo eseguire un programma di prova client ActiveX in un altro progetto, la cattura di errori nel progetto server deve essere impostata a *Break in Class Module.* Questo aiuterà a localizzare accuratamente il problema.
Le opzioni *Compile On Demand* e *Background Compile* permettono di ridurre i tempi di compilazione abilitando rispettivamente la compilazione solo su richiesta e in background.

La scheda Docking

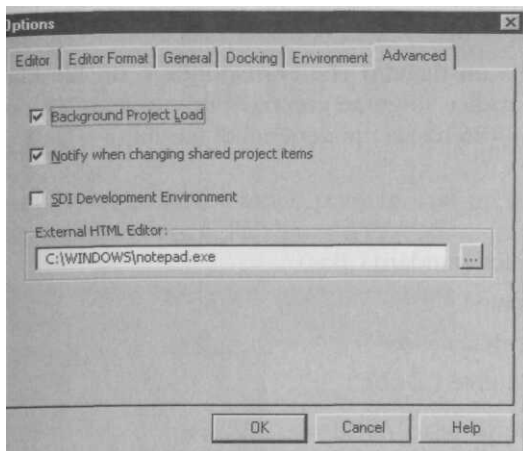
La scheda *Docking,* mostrata nella Figura 2.9, serve a determinare quali elementi dell'interfaccia utente di Visual Basic possono venire "ancorati", ossia attaccati ad una posizione specifica invece che lasciati mobili.

Figura 2.9
La scheda Docking permette di impostare quali elementi dell'interfaccia utente di Visual Basìcpossono venire ancorati.

La scheda Advanced

La scheda *Advanced,* mostrata nella Figura 2.10, permette di abilitare il caricamento in background del progetto e la notifica di quando i moduli condivisi del progetto vengono modificati.
Questa scheda abilita anche il passaggio dell'IDE di Visual Basic fra l'interfaccia a documenti multipli (in inglese, *Multiple Document Interface,* abbreviato in MDI) e l'interfaccia a documento singolo (in inglese, *Single Document Interface,* abbreviato in SDÌ).

// caricamento del codice in background accresce la velocità di caricamento di Visual Basic.

I file di progetto di Visual Basic contengono solo i riferimenti alla posizione dei moduli inclusi nel progetto, e non i moduli stessi. Due o più progetti possono contenere riferimenti allo stesso modulo. Se non si sta attenti, modificando un modulo di un progetto inavvertitamente si modificano anche altri progetti. Se si imposta

Figura2.10
la scheda Advanced serve a impostare il caricamento in background del progetto, la notifica di modifica delle voci di progetto condivise, ed'ambiente di sviluppo SDÌ o MDI.

Notify When Changing Shared Project Items, si verrà avvertiti quando si modifica un elemento di progetto condiviso (per esempio un form o un modulo) e si tenta di salvarlo.

Più progetti possono condividere le stesse voci. Le voci condivise vengono caricate in memoria e ogni progetto ha la sua copia. Se si modifica una voce condivisa di un progetto, gli altri progetti mantengono la copia della voce che era stata caricata fino a quando si salvano i progetti. Allora, l'ultimo progetto che viene salvato determina ciò che si troverà nel file condiviso.

Quando viene impostata l'opzione *Notify When Changing Shared Project Items,* viene chiesto se si vogliono sincronizzare tutte le copie di quell'elemento prima di procedere al salvataggio.

Le prime versioni di Visual Basic usavano un'interfaccia utente a documento singolo, nel senso che si poteva aprire solamente un progetto per volta. Di default, VB6 usa un'interfaccia a documenti multipli.

Volendo, si può far passare VB dall'interfaccia SDÌ alla MDI abilitando o disabilitando la casella di opzione *SDI Development Environment* presente su questa scheda.

Nelle precedenti versioni di VB, si impostavano alcune opzioni di compilazione riferite ad un progetto specifico (per esempio, gli argomenti della riga di comando) mediante la finestra di dialogo *Options.* Ciò era possibile perché si poteva caricare solamente un progetto per volta. Adesso si applica questo tipo di impostazione nella finestra di dialogo *Project Properties,* a cui si accede dal menu *Project.* Per ulteriori informazioni, si veda il paragrafo "La compilazione degli eseguibili" nel seguito di questo capitolo.

La *scheda* Advanced *serve anche a selezionare l'editor HTML esterno che si userà per le applicazioni Web.*

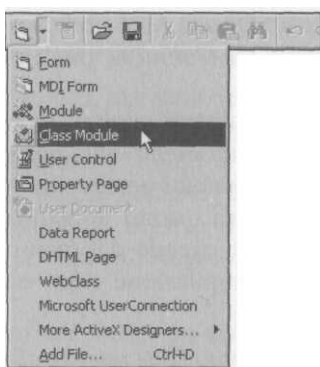
Aggiunta di diversi moduli a un progetto

Un modulo Visual Basic è un oggetto che corrisponde a un file che può venire incluso in un progetto. Il codice sorgente grezzo di un progetto VB consiste primariamente in file di moduli. VB6 ha sei tipi generici di moduli:

- File di forni (.Frm)
- File di classe (.Cls)
- File di modulo di codice standard (.Bas)
- File di controllo d'utente (.Ctl)
- File di pagina di proprietà (.Pag)
- File di documento d'utente (.Dob)

I file di modulo sono file di testo ASCII che contengono del codice incapsulato entro il modulo. Se i moduli comprendono dei controlli, le descrizioni dei controlli vengono incluse nel file.
Alcuni tipi di moduli sono anche collegati a dei file binari che contengono dei dati sull'aspetto del modulo: per esempio, i file .Frm possono essere collegati ai file .Frx che hanno lo stesso nome a parte l'estensione del file. I file .Frx contengono informazioni binarie sull'aspetto del forni, mentre il file .Frm contiene codice di modulo, riferito ai controlli utilizzati sul forni, e informazioni sulle impostazioni delle proprietà dei controlli utilizzati e del form stesso. Si troveranno ulteriori informazioni sulla struttura interna dei file .Frm nel Capitolo 19, nel paragrafo "La vita segreta dei form".
Ci sono vari modi per aggiungere moduli di diverso tipo ad un progetto Visual Basic:

- Si può usare la casella di riepilogo a discesa sotto il pulsante *Add Form* sulla barra degli strumenti, come mostrato nella Figura 2.11.

Figura 2.11 *La casella di riepilogo a discesa sotto il pulsante Add Form della toolbar serve ad aggiungere nuovi moduli a un progetto.*

- Si possono aggiungere nuovi moduli mediante il menu *Project.*
- Si può scegliere *Add* dal menu di scelta rapida nel Project Explorer.

Quando si aggiunge un modulo a un progetto, quel modulo viene automaticamente aggiunto all'elenco dei file gerarchicamente associati al progetto nel Project Explorer.

Il menu Edit

Il menu *Edit* in Visual Basic 6 è molto potente, con un numero davvero elevato di opzioni, come si può vedere nella Figura 2.12.

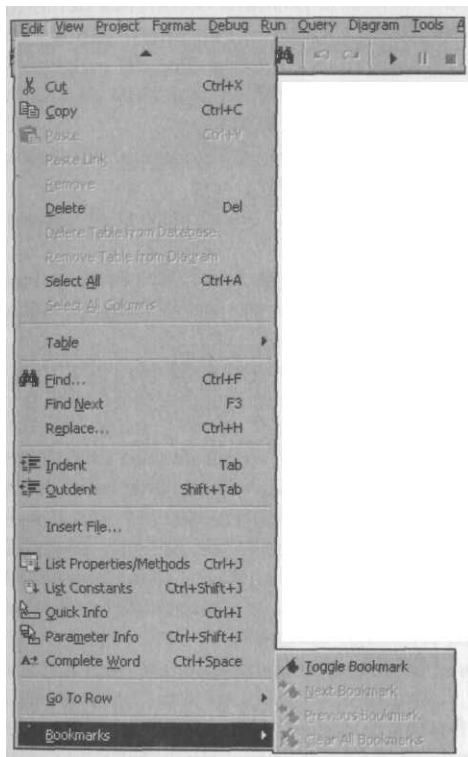
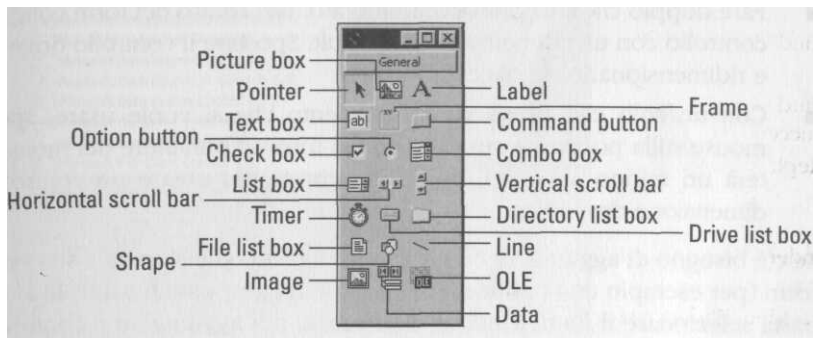
Figura 2.12
// menu Edit adesso contiene parecchie opzioni che possono facilitare l'attività di programmazione.

La Tabella 2.3 elenca la maggior parte delle voci del menu *Edit* e descrive il loro scopo. Oltre alle voci della tabella, il menu comprende gli strumenti standard per la modifica dei testi come *Cut [Taglia], Copy [Copia], Delete [Elimina]* e *Paste [Incolla]*. Alcune altre voci saranno disponibili quando si lavorerà con i database.

Tabella 2.3 *Il menu Edit.*

Voce del menu Edit	Scopo
Undo [Annulla]	Rovescia la precedente azione di modifica. Sono disponibili venti livelli di annullamento e ripetizione.
Redo [Ripeti]	Ripristina la precedente modifica di testo se non sono avvenute altre azioni dall'ultimo annullamento.
Find [Trova]	Cerca, in un ambito di ricerca specificato, il testo specificato nella casella *Find What* della finestra di dialogo *Find.*
Find Next [Trova successivo]	Trova e seleziona la prossima ocorrenza del testo specificato nella casella *Find What* della finestra di dialogo *Find.*
Replace [Sostituisci]	Cerca il testo specificato nel codice del progetto e lo sostituisce con il nuovo testo specificato nella casella *Replace With* della finestra di dialogo *Replace.*
Indent [Aumenta rientro]	Fa scorrere tutte le righe selezionate al successivo arresto di tabulazione.
Outdent [Riduci rientro]	Fa scorrere tutte le righe selezionate al precedente arresto di tabulazione.
Insert File [Inserisci file]	Apre la finestra di dialogo *Insert File* per poter inserire il testo di un file esistente alla posizione corrente nella finestra di codice.
List Properties/Methods [Elenca proprietà/metodi]	Apre una casella di riepilogo a discesa nella finestra di codice che contiene le proprietà e i metodi disponibili per un oggetto appena viene introdotto nella finestra di codice seguito dall'operatore punto (.). Viene attivato dall'opzione *Auto List Members* della scheda *Editar* della finestra di dialogo *Options.*
List Constants [Elenca costanti]	Apre una casella di riepilogo a discesa nella finestra di codice che contiene le costanti valide per una proprietà introdotta nella finestra di codice seguita dal segno uguale (=). Viene attivato dall'opzione *Auto List Members* della scheda *Editar* della finestra di dialogo *Options.*
Quick Info [Informazioni rapide]	Fornisce la sintassi per una variabile, una funzione, un'istruzione, un metodo, o una procedura che siano stati selezionati nella finestra di codice. Attivata dall'opzione *Auto Quick Info* nella scheda *Editar* della finestra di dialogo *Options.*
Parameter Info Informazioni parametri]	Visualizza informazioni sui parametri della prima funzione o istruzione inserita.
Complete Word [Completa parola]	Completa la parola che si sta digitando una volta che si sono inseriti abbastanza caratteri da permettere a Visual Basic di identificare la parola voluta.
Bookmarks [Segnalibri]	Visualizza un menu che si può usare per creare o togliere dei segnalibri nella finestra di codice, spostarsi al segnalibro successivo o al precedente, o togliere tutti i segnalibri. Quando si imposta un segnalibro, la sua posizione viene rappresentata sul margine sinistro della finestra di codice da un ovale blu.

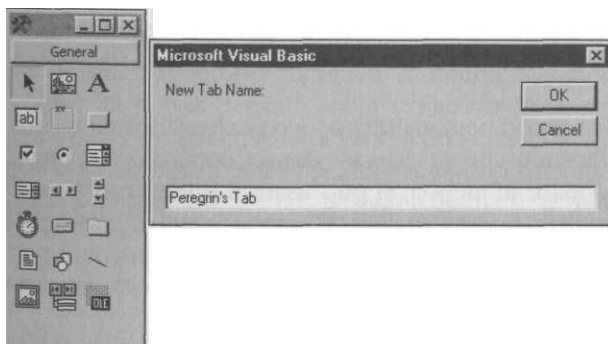
La Toolbox

La Toolbox serve ad aggiungere controlli ai contenitori come i form mentre si è in modalità progettazione. La Figura 2.13 mostra la Toolbox con l'insieme di strumenti dell'edizione Enterprise appena installata.

Figura 2.13 *La Toolbox di Visual Basic serve ad aggiungere controlli (come i controlli d'utente) ai form e ad altri contenitori.*

La Toolbox comprende dei controlli standard che ci sono sempre. Questi controlli standard (talvolta chiamati controlli intrinseci) sono lo strumento puntatore, il controllo casella di testo, il pulsante di comando, il pulsante di opzione, il controllo immagine, e così via. In realtà il puntatore non rappresenta un controllo; viene usato nell'ambiente di progettazione per selezionare gli oggetti.

I controlli personalizzati sono quelli che esistono in file separati con estensione .Ocx e che possono venire aggiunti alla Toolbox quando serve.

Alla Toolbox si possono aggiungere delle schede *(tab),* che possono essere utilizzate per organizzare i controlli contenuti nella casella degli strumenti. La Figura 2.14 mostra il processo di aggiunta di una scheda alla Toolbox. Per accedere alla finestra di dialogo *New Tab Name,* fate clic con il tasto destro su una scheda esistente e scegliete *Add* dal menu di scelta rapida. Per spostarsi fra le schede della Toolbox, basta fare clic sulle "linguette" che identificano le schede stesse.

Figura 2.14 *Si possono organizzare facilmente i controlli ActiveX aggiungendo schede personalizzate alla Toolbox*

Aggiunta di controlli ai form

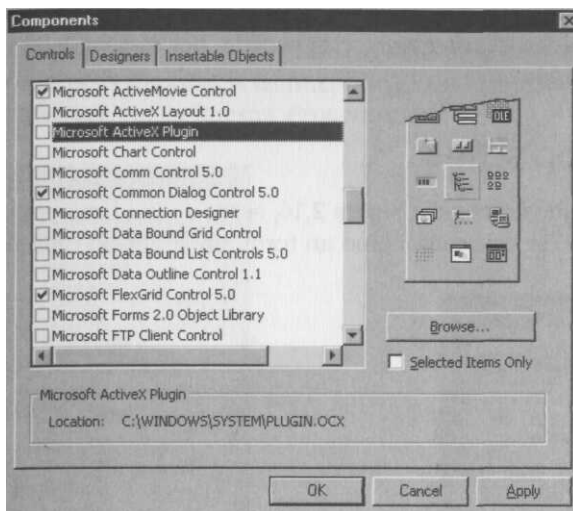
Ci sono due modi per aggiungere un controllo a un form (o a un altro contenitore). Dapprima si seleziona il form a cui si vuole aggiungere il controllo. Poi, si può seguire una delle due strade seguenti:

- Fare doppio clic sul controllo desiderato. Nel centro del form comparirà un controllo con una dimensione di default. Spostare il controllo dove si vuole e ridimensionarlo, se necessario.
- Con un solo clic, scegliere lo strumento che si vuole usare, spostare il mouse sulla posizione appropriata del form (il puntatore del mouse diventerà un mirino a croce), quindi trascinare per creare un controllo della dimensione desiderata.

Se c'è bisogno di aggiungere controlli a un oggetto che si trova a sua volta sopra un form (per esempio una cornice o f*rame,* che raggruppa dei controlli al suo interno) basta selezionare il frame (invece del form) e poi aggiungere il controllo mediante uno dei due metodi appena descritti. Si noti che, quando si sposta il frame sul form, i controlli contenuti in tale frame si spostano solidalmente.
Per creare rapidamente una *matrice* o *array* di controlli *(control array),* per esempio, una matrice di pulsanti di opzione (detti anche *radio button),* rilasciare il primo pulsante di opzione sul form dove si desidera che risieda. Impostare le sue proprietà (l'impostazione delle proprietà verrà discussa più avanti in questo capitolo nel paragrafo "La finestra Properties"). Poi, selezionarlo, copiarlo con i comandi del menu *Edit,* poi scegliere *Paste* dal menu *Edit.* Una finestra di dialogo chiede se si vuole creare una matrice. Scegliere *Yes* e VB automaticamente gli attribuisce un indice di matrice. Continuare a incollare tanti pulsanti di opzione quanti ne servono. (Alternativamente, per creare una matrice, si possono usare le proprietà *.Name* dei pulsanti di opzione.)

Aggiunta di componenti alla Toolbox

Per aggiungere un componente alla Toolbox, scegliete *Componente* dal menu *Projects* oppure premete Ctrl+T. Si vedrà la finestra di dialogo *Components* mostrata nella Figura 2.15.
Per aggiungere un controllo personalizzato, occorre trovarlo sull'elenco e impostare la relativa casella facendo clic su di essa. (Se un controllo personalizzato è stato installato ma non appare in elenco, si può usare il pulsante *Browse* per trovarlo.) Nella parte inferiore della finestra di dialogo compariranno il nome e il percorso del controllo. Se poi si fa clic su *OK,* la finestra di dialogo scompare, il nuovo controllo viene aggiunto alla Toolbox, e la Toolbox si ridimensiona automaticamente per fare posto al nuovo controllo.

Figura 2.15
La finestra di dialogo Components serve ad aggiungere controlli personalizzati alla Toolbox.

Gli oggetti inseribili

Qualcuno probabilmente avrà notato la scheda *Insertable Objects* nella finestra di dialogo *Components* presentata nella Figura 2.15. Per *oggetti inseribili* si intendono gli oggetti OLE esposti, per esempio, un foglio elettronico Excel o un documento Word. Una volta che l'oggetto è stato aggiunto alla casella di riepilogo nella finestra di dialogo, lo si può aggiungere alla Toolbox, e poi aggiungerlo a qualsiasi form VB nel modo normale. Questo è un modo straordinariamente comodo di aggiungere la potenza di altre applicazioni ai propri progetti VB; lo tratteremo in dettaglio nella Parte V del libro, "Utilizzo di ActiyeX".

I designer

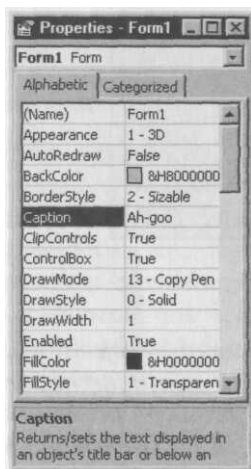
I *designer* ActiveX sono un modo di estendere l'ambiente Visual Basic in un modo personalizzato. Nell'ambiente VB sono incorporati molti designer: per esempio, il designer dei form e quello dei controlli d'utente. Questi designer incorporati forniscono interfacce visuali per le operazioni che altrimenti richiederebbero una grande quantità di codice. Li si può pensare anche come un'estensione dei fogli di proprietà personalizzati. I designer ActiveX producono classi da cui si possono creare oggetti. Queste classi appaiono nella finestra *Project,* proprio come le classi di form. La finestra di progettazione di un designer ActiveX è pienamente integrata nell'ambiente di sviluppo. La si può dimensionare e disporre proprio allo stesso nodo delle finestre di progettazione incorporate, come il designer dei form.

Le edizioni Professional e Enterprise di Visual Basic comprendono il Software DevelopmentKit (SDK) dei designer ActiveX, che serve a creare nuovi designer ActiveX da usare con VisualBasic. L'SDK dei designer ActiveX ha bisogno di un compilatore C++, come il Microsoft Visual C++. Non si possono scrivere designer ActiveX usando solo Visual Basic. Per ulteriori informazioni, si veda la Pane VII, "Estensione dell'ambiente Visual Basic".

Nel Toolbox non ci sono designer. Una volta che si è aggiunto un designer attraverso la finestra di dialogo *Components,* vi si accede per mezzo della voce *ActiveX Designers* del menu *Project.*

La finestra Properties

La finestra *Properties,* mostrata nella Figura 2.16, serve a visualizzare *e* modificare le proprietà di un oggetto selezionato, come un form, un controllo o un modulo.

Figura 2. 16 *La finestra Properties serve a impostare leproprietà infase di progettazione. Qui vengono mostrate le proprietà di un controllo Label.*

Questa finestra ha più parti. In cima c'è l'*ObjectBox,* che visualizza il nome e il tipo dell'oggetto. (Nella Figura 2.16, è selezionato un forni chiamato *Form1.)* Facendo clic sulla freccia rivolta in basso sulla destra della casella si può selezionare qualunque oggetto (per esempio i controlli) che sia disponibile sul form selezionato.
Il lungo elenco di voci sul lato sinistro della Figura 2.16 contiene le proprietà dell'oggetto. (La lunghezza dell'elenco dipende da quante proprietà ha l'oggetto: in qualche caso l'elenco è molto breve.) La colonna sulla destra mostra quali siano al momento le impostazioni delle proprietà.

Modifica delle proprietà degli oggetti

Per modificare l'impostazione di una proprietà, selezionate la proprietà e poi fate clic sulla casella corrispondente (la "casella di impostazione") nella colonna a destra della finestra *Properties.* Se la proprietà assume un valore stringa o intero, nella casella di impostazione comparirà un cursore lampeggiante. Se la proprietà può contenere un determinato elenco di valori, sulla destra della casella di impostazione comparirà un pulsante che mostra una freccia a discesa oppure dei puntini (...). Inoltre, a molte impostazioni di proprietà sono collegate finestre di dialogo personalizzate (cosa indicata sempre da puntini). Molto spesso, dalla proprietà *Custom* si accede a finestre di dialogo personalizzate. Per accedere a queste impostazioni e selezionare ciò che si vuole, bisogna fare clic sul pulsante contenente la freccia rivolta in basso o i puntini.

Suddividere in categorie le proprietà

Come si vede nella Figura 2.17, si può scegliere di visualizzare le proprietà di un oggetto ordinate per categoria. A tal fine; bisogna selezionare la scheda *Categorized* (invece di *Alphabetic)* nella finestra *Properties.*

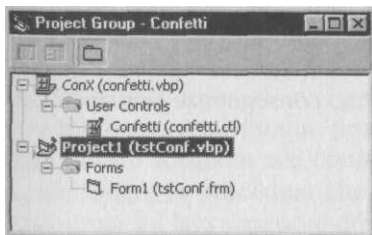
Figura 2.17 *Si può scegliere di visualizzare le proprietà ordinate per categoria.*

Ogni categoria principale può venire contratta o espansa per mostrare le proprietà della categoria. Queste possibilità sono indicate dal segno più (che sta per "qui c'è dell'altro") e dal segno meno (che significa "contrai per non vedere queste voci") sul lato sinistro della finestra *Properties* (Figura 2.17).

Il Project Explorer

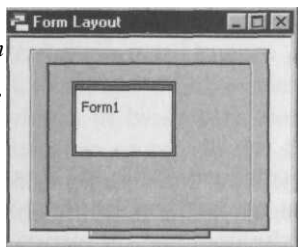
Un file di gruppo Visual Basic (.Vbg) è semplicemente un file ASCII contenente un elenco di tutti i progetti Visual Basic associati a un dato gruppo. Analogamente, un file di progetto Visual Basic (.Vbp) è un file che contiene un elenco di tutti i file associati a un dato progetto.

Il Project Explorer è una finestra che fornisce una vista gerarchica dei gruppi, dei progetti e dei moduli sorgente associati ai progetti, come si può vedere nella Figura 2.18. Il Project Explorer fornisce un modo facile di navigare fra le parti dei progetti e dei gruppi di progetti.

Figura 2.18 *Il Project Explorer serve a navigare fra i diversi moduli di un progetto e fra i diversi progetti di un gruppo.*

Quando si lancia Visual Basic usando un file di gruppo o un file di progetto (o si apre un gruppo esistente o un progetto esistente da VB), i file sorgente che sono' associati al gruppo o al progetto vengono visualizzati nella finestra Project Explorer. Quando si aggiungono o tolgono dei file (per esempio dei form, dei moduli, e così via) a o da un progetto, queste modifiche si riflettono nel Project Explorer.
Nel Project Explorer appaiono due "nomi" per ogni form o modulo. Il nome sulla sinistra è quello della proprietà che è stata assegnata al form quando il programmatore lo ha creato, per esempio, dlgAbout. Il nome sulla destra (fra parentesi) è invece quello del file sorgente con cui il modulo è stato salvato, per esempio, Aboutbox.frm.
I pulsanti della barra degli strumenti in cima al Project Explorer (Figura 2.18) servono a vedere il codice di ogni modulo sorgente, a vedere un modulo come un form in un designer, e a contrarre (o espandere) l'albero dell'Explorer. Inoltre, si può usare un ampio menu di scelta rapida, che si modifica dinamicamente a seconda di ciò che è selezionato, per manipolare rapidamente l'oggetto selezionato.

La finestra Form Layout

La finestra *Form Layout* serve a posizionare i form sullo schermo. Come si vede nella Figura 2.19, quando si passa il puntatore del mouse sopra un form nella finestra *Form Layout,* esso si trasforma in un'icona di trascinamento. A questo punto si può riposizionare il form trascinandolo.

Figura 2.19
La finestra Form Layout serve a riposizionare i form sullo schermo.

Nell'IDE di Visual Basic 6, i form sono bloccati nell'angolo superiore sinistro dello spazio di lavoro del designer. Per quanto si possa ridimensionarli nel designer (trascinando il bordo inferiore o il bordo destro del form), non si può spostarli rispetto allo schermo senza usare la finestra Form Layout.

Normalmente, in fase di esecuzione la posizione dei form sullo schermo viene modificata dalle azioni dell'utente (per esempio, quando l'utente ridimensiona una finestra) e dal codice quando il programma viene eseguito. Perciò, le modifiche apportate nella finestra *Form Layout* hanno conseguenze principalmente sulle posizioni iniziali dei form.

Il menu Format

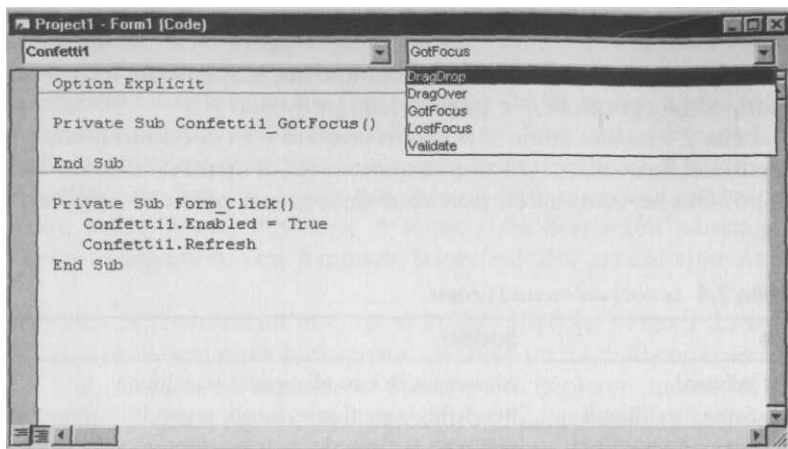
Le voci del menu *Format* servono a manipolare l'aspetto dei forni e degli altri contenitori, come i controlli e le pagine delle proprietà.
La Tabella 2.4 mostra come si possono usare le voci di questo menu per modificare l'aspetto del forni attivo. (Alcune di queste voci di menu richiamano delle finestre di dialogo, che servono a selezionare delle opzioni prima di applicare la formattazione.)

Tabella 2.4 *Le voci del menu Format.*

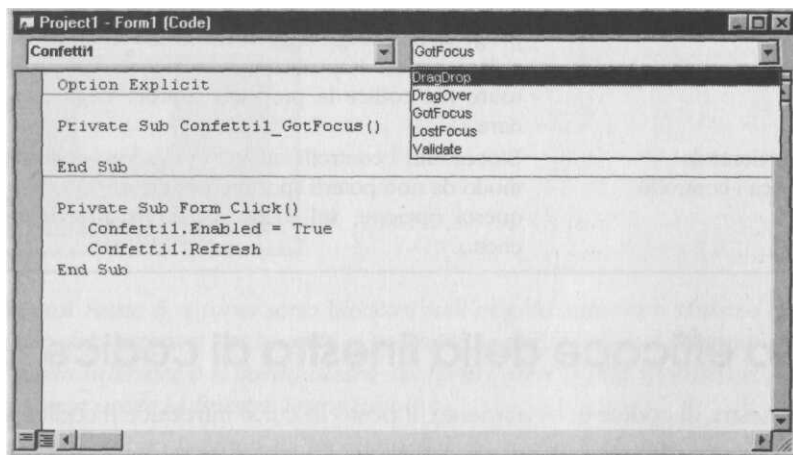
Voce	Scopo
Align [Allinea]	Allinea tra di loro gli oggetti selezionati.
Make Same Size [Rendi della stessa dimensione]	Rende gli oggetti selezionati, secondo la direzione scelta (o le direzioni scelte), della stessa dimensione dell'oggetto selezionato per ultimo.
Size to Grid [Dimensiona alla griglia]	Regola l'altezza e la larghezza dell'oggetto selezionato in modo da raggiungere le linee di griglia più vicine nel form.
Horizontal Spacing [Spaziatura orizzontale]	Modifica lo spazio orizzontale fra gli oggetti selezionati.
Vertical Spacing [Spaziatura verticale]	Modifica lo spazio verticale fra gli oggetti selezionati.
Center in Form [Centra nel form]	Centra sul form verticalmente o orizzontalmente gli oggetti selezionati.
Order [Ordinamento]	Modifica l'ordinamento dal posteriore all'anteriore in fase di progettazione degli oggetti selezionati di un form. (Per manipolare questa impostazione in fase di esecuzione, bisogna usare nel codice la proprietà ZOrder degli oggetti da ordinare.)
Lock Controls [Blocca i controlli]	Blocca tutti i controlli sul form nella loro posizione attuale in modo da non poterli spostare per errore. Quando è impostata questa opzione, sul menu viene visualizzata un'icona lucchetto.

Uso efficace della finestra di codice

La finestra di codice è, ovviamente, il posto in cui si introduce il codice Visual Basic. (Si veda il Capitolo 3 per avere informazioni su come il codice interagisce con gli oggetti e il Capitolo 4 per i dettagli di programmazione in Visual Basic.) La Figura 2.20 mostra una finestra di codice per un form con un pulsante di comando chiamato *cmdOK* collocato sul form.
Per spostarsi rapidamente a qualunque procedura di qualunque controllo disponibile su un form, si può scegliere tale controllo dalla casella di riepilogo a discesa *Object* e poi selezionare la procedura che si desidera dalla casella di riepilogo a discesa *Proc,* mostrata sul lato superiore destro della Figura 2.20. Nelle procedure si naviga come nella finestra di codice di un modulo.

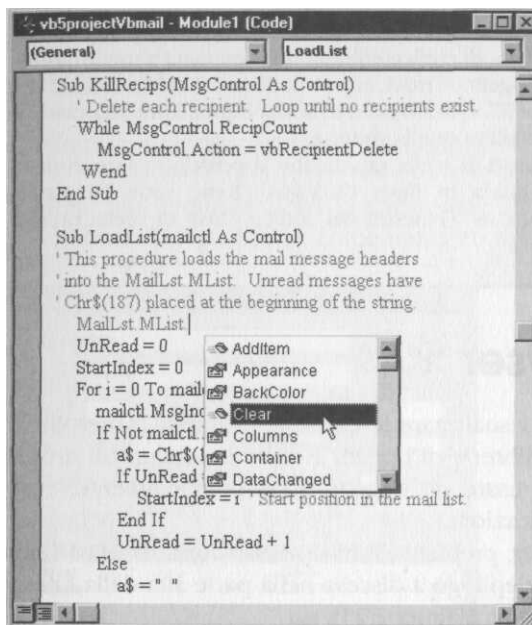
Figura 2.20
Questa finestra di codice di VB mostra la casella di riepilogo a discesa Procedure aperta.

Intelligenza artificiale

No, Visual Basic 6 in realtà non è dotato di un motore di intelligenza artificiale, per quanto gli strumenti che sono sempre disponibili nella finestra di codice con un clic del pulsante destro del mouse possano sembrare maledettamente furbi. Come si vede nella Figura 2.21, è disponibile un esteso arsenale di ausili alla codifica.

Figura 2.21
Si può accedere ad alcuni ausili di programmazione assai intelligenti solofacendo clic col tasto destro del mouse.

In sostanza, se si prende l'abitudine di usare questi strumenti, non c'è più bisogno né di ricordarsi la sintassi né di andarla a controllare su un manuale. Qualche bisbetico potrà forse non apprezzarlo, e potrà persino andare in giro brontolando, "Mi ricordo quando. . .". Per conto mio, non ho intenzione di tornare a fare il dattilografo e qualunque aiuto mi possa dare un ambiente di sviluppo, se mi fa risparmiare tempo, è sempre benvenuto.

Gli ausili di programmazione disponibili sono *List Properties/Methods, List Constants, Quick lnfo, Parameter Info* e *Complete Word,* come segue:

- *ListProperties/Methods* visualizza una casella di riepilogo a discesa di tutte le proprietà e di tutti i metodi disponibili per l'oggetto al punto di inserimento, come si vede nella Figura 2.22.

Figura 2.22
List Properties/ Methods serve a trovare rapidamente le proprietà e i metodi validi per un oggetto.

- *List Constants* visualizza una casella di riepilogo a discesa di tutte le costanti valide per una proprietà che è stata introdotta nella finestra di codice.
- *Quick Info* fornisce la sintassi di una variabile, una funzione, un'istruzione, un metodo o una procedura selezionati.
- *Parameter Info* visualizza una finestra a comparsa che contiene informazioni sui parametri di una funzione o di un'istruzione.
- *Complete Word* completa il resto della parola che si sta digitando quando si sono introdotti abbastanza caratteri da permettere a Visual Basic di identificare la parola voluta. La maggior parte delle volte, la indovina. Mi aspetto che, se riusciranno a perfezionarlo ancora un po', potrò lasciare il computer acceso, farmi una buona dormita notturna e tornare per trovare il mio lavoro di programmazione fatto senza di me!

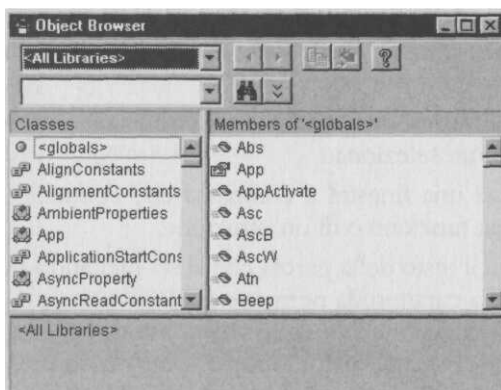
i tasti di scelta rapida e la finestra di codice

Molti tasti di scelta rapida (in inglese, *shortcut keys)* possono facilitare la vita quando si sta usando la finestra di codice. Per esempio, ci si può spostare su o giù di una procedura per volta premendo Ctrl in combinazione con i tasti freccia su e freccia giù, rispettivamente.
Un altro tasto di scelta rapida che è un salvavita in un grande progetto è Maiusc+F2. Se un'istruzione nella finestra di codice effettua una chiamata a una routine che non si sa in quale modulo del progetto si trovi, si può collocare il cursore entro il nome della routine e premere Maiusc+F2. Questo tasto di scelta rapida apre automaticamente una finestra di codice che visualizza quella routine.
Per avere un elenco dei tasti di scelta rapida che si possono usare in una finestra di codice, guardate nella guida in linea di Visual Basic sotto l'argomento "Code Window—Keyboard Shortcuts" (Finestra del codice - tasti di scelta rapida nella versione italiana).

L'Object Browser

L'Object Browser serve a visualizzare le classi, le proprietà, i metodi, gli eventi e le costanti disponibili nelle librerie di oggetti e nelle procedure di un progetto. Lo si può usare per trovare e usare gli oggetti creati personalmente, come anche gli oggetti tratti da altre applicazioni.
Per aprire PObject Browser, premete F2 o scegliete *Object Browser* dal menu *View.* Si può usare la casella di riepilogo a discesa nella parte alta della finestra per visualizzare una particolare libreria (Figura 2.23).

Figura 2.23

L'Object Browser serve a visualizzare le classi e i loro membri.

Si può usare la casella di riepilogo a discesa che sta sotto l'elenco delle librerie di oggetti per introdurre il testo da trovare (si può anche selezionarlo da un elenco).
Il pannello *Classes,* sulla sinistra nella Figura 2.23, visualizza tutte le classi disponibili della libreria selezionata. Se per una classe è stato scritto del codice, quella classe appare in grassetto. L'elenco inizia sempre con *<globals>,* un elenco di membri accessibili globalmente.

Ti pannello *Members* (sulla destra in Figura 2.23) visualizza gli elementi della classe selezionata nel pannello *Classes* ordinati per gruppo e poi in ordine alfabetico all'interno di ciascun gruppo. I metodi, le proprietà, gli eventi, o le costanti per cui è stato scritto del codice appaiono in grassetto.
Il pannello *Details* (sul fondo nella Figura 2.23) mostra la definizione di un membro selezionato.

Il Menu Editor

Si possono facilmente aggiungere menu a qualunque form di un progetto mediante il Menu Editor di Visual Basic. Per farlo, bisogna selezionare il form a cui si vuole aggiungere il menu, poi premere Ctrl+E o selezionare *Menu Editor* dal menu *Tools*. Comparirà la finestra del Menu Editor, come mostrato nella Figura 2.24.

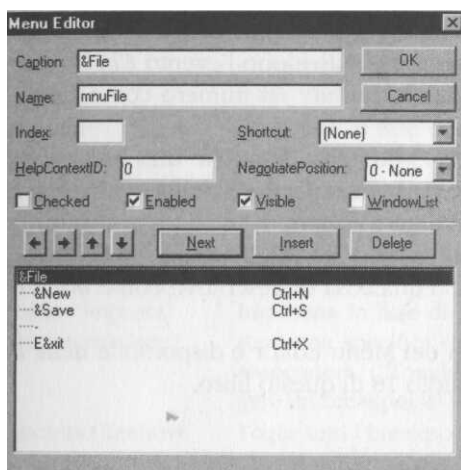
Figura 2.24
Il Menu Editor serve ad aggiungere menu ai form.

Come si vede nella figura, sono state aggiunte alcune voci al Menu Editor: un titolo di menu *File,* e tre voci di menu sotto quel titolo *(New, Save* e *Exit)*. La Figura 2.25 mostra il form con questo mini-menu aperto.

Figura 2.25
Ecco un menu creato con il Menu Editor.

Come si vede nella Figura 2.24, le voci per il titolo del menu, *File,* sono senza problemi. Dapprima, è stato digitato il nome del titolo nella casella di testo *Caption.* Notate che è stata aggiunta una e commerciale (&) prima della *F.* Questo crea un tasto di scelta rapida, nel senso che l'utente del programma può accedere a questo

menu usando la combinazione di tasti Alt+F. Successivamente, è stato dato un nome al titolo del menu, *mnuFile,* proprio come si darebbe un nome a qualunque controllo Visual Basic. Poi, è stato fatto clic sul pulsante *Next* per introdurre quel titolo di menu e spostarsi avanti alla prima voce del menu.
Le voci di menu mostrate in Figura 2.24 *(New, Save* ed *Exit)* sono tutte precedute da righe punteggiate, le quali indicano che si tratta di sotto-voci del menu. Per rendere una voce sottovoce della precedente, nel Menu Editor, bisogna selezionare la voce nella casella di riepilogo sul fondo della finestra e premere il pulsante freccia a destra. Visual Basic permette fino a quattro livelli di sottomenu. Per spostare una voce di menu su di un livello, si preme il pulsante freccia a sinistra.
Mediante le appropriate caselle di controllo nel Menu Editor, si può fare in modo che una voce di menu appaia con un segno di spunta, si può abilitare o disabilitare qualunque voce di menu, o si può renderla invisibile o visibile. Si può anche assegnare un tasto di scelta rapida a ogni voce di menu (per esempio, la Figura 2.24 mostra la voce di menu *New* con il tasto di scelta rapida Ctrl+N).
Si può anche impostare una matrice di controlli di voci di menu (cioè, un array di voci di menu con lo stesso nome che condividono l'evento *Click)* grazie alla casella di testo *Index,* come anche si può impostare un numero come "help ID" per ogni voce di menu.
Il menu mostrato nella Figura 2.25 comprende anche una barra separatrice fra le voci di menu *Save* ed *Exit.* Per creare un separatore nel Menu Editor, bisogna introdurre un trattino (-) nella casella di testo *Caption,* poi dare qualunque nome si desideri. Sebbene in realtà non abbia importanza il nome che viene dato a questi separatori, spesso li si chiama con qualcosa di descrittivo come *mnuSep1, mnuSep2* e simili.
Una documentazione dettagliata del Menu Editor è disponibile nella *Programmer's Guide* di Visual Basic e nel Capitolo 18 di questo libro.

Gli strumenti di debug

Quando si fa il debug di un'applicazione si è, naturalmente, alla ricerca di fonti di errori. Visual Basic offre vari strumenti che possono facilitare questo processo spesso tedioso. (Non occorre dirlo, ma sarebbe meglio creare programmi che non contengano bachi fin dall'inizio. Essendo praticamente impossibile ottenere questo obiettivo, i programmi dovrebbero almeno intercettare tutti gli errori che si verificano e dire quali sono!)
In primo luogo, Visual Basic può operare in tre modalità: modalità progettazione *(design mode),* modalità esecuzione *(run mode), e* modalità interruzione *(break mode).* Queste modalità sono semplicemente quello che ci si aspetterebbe. La modalità progettazione serve per la maggior parte del lavoro di creazione di un'applicazione. La modalità esecuzione è per eseguire l'applicazione nell'ambiente di debug di VB in modo da poter interagire con il programma nel modo in cui farebbe un utente (eventualmente con alcune differenze dovute al fatto che non viene eseguita la versione compilata dell'applicazione). Si entra in modalità interruzione quando l'esecuzione di un programma viene sospesa. Si può accedere ad

ognuna di queste tre modalità attaverso il menu *Run o* con l'appropriato pulsante della barra degli strumenti.

Si accede agli strumenti di debug di Visual Basic tramite il menu *Debug.* La Tabella 2 5 elenca le opzioni disponibili in questo menu. Solamente i breakpoint e i watch possono essere impostati in fase di progettazione; tutti gli altri strumenti di debug funzionano solo in modalità interruzione.

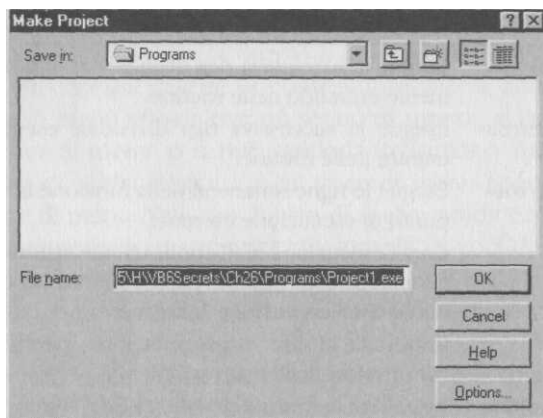
Tabella 2.5 *Strumenti di debug.*

Strumento di debug	Descrizione
Step Into (Esegui istruzione)	Esegue la successiva riga di codice eseguibile, eventualmente entrando nelle routine.
Step Over (Esegui istruzione/routine)	Esegue la successiva riga di codice eseguibile, senza entrare nelle routine.
Step Out (Esci da istruzione/routine)	Esegue le righe rimanenti della funzione in cui si trova il punto di esecuzione corrente.
Run to Cursor (Esegui fino al cursore)	Con l'esecuzione dell'applicazione arrestata, permette di collocare il cursore più in giù nel codice dove si vuole che l'esecuzione si arresti.
Add Watch (Aggiungi espressione di controllo)	Impostata in fase di progettazione; permette di osservare i valori delle espressioni.
Edit Watch (Modifica espressione di controllo)	Visualizza la finestra di dialogo *Edit Watch,* in cui si può modificare o togliere un'espressione di watch.
Quick Watch (Controllo immediato)	Mostra il valore corrente di un'espressione.
Toggle Breakpoints (Imposta/rimuovi punto di interruzione)	Impostata in fase di progettazione; permette di impostare una specifica riga di codice dove l'esecuzione si sospenderà. ("Toggle" si riferisce a togliere o aggiungere un breakpoint).
Clear Ali Breakpoints (Rimuovi punti di interruzione)	Toglie tutti i breakpoint del progetto.
Set Next Statement (Imposta istruzione successiva)	Permette di impostare la prossima istruzione di codice da eseguire, anche se è già stata eseguita.
Show Next Statement (Mostra istruzione successiva)	Permette di spostare il cursore alla successiva riga di codice che verrà eseguita quando il programma continuerà l'esecuzione.

Si possono anche selezionare tre opzioni di cattura di errori (*Break on Ali Errors, Break in Class Module* e *Break on Unhandled Errors)* nella scheda *General* della finestra di dialogo *Options* (scegliere *Tools/Options).* Inoltre, si può fare clic con il tasto destro del mouse su una finestra di codice aperta per avere accesso immediato a queste tre opzioni. Il procedimento di debug è discusso in dettaglio nel Capitolo 15, "Gestione degli errori".

La compilazione degli eseguibili

Per compilare un eseguibile in Visual Basic, bisogna scegliere *Make* dal menu *File*. Comparirà una finestra di dialogo come quella mostrata nella Figura 2.26. Come si vede nella figura, si può salvare l'eseguibile sotto qualunque nome e lo si può collocare nella cartella di default o in una cartella di propria scelta spostandosi in quella cartella.

Figura 2.26
La finestra di dialogo Make Project serve a creare un eseguibile.

Ciò che è veramente interessante della finestra di dialogo *Make Project* è il pulsante *Options*. Facendo clic su questo pulsante si visualizza la finestra di dialogo *Project Properties* mostrata in Figura 2.27.
Qui si possono specificare le informazioni di versione sull'eseguibile, tra cui i numeri di versione primario (in inglese, *major),* secondario (in inglese, *minor), e* di revisione (in inglese, *revision).* Se si abilita la casella *Auto Increment,* VB incrementa automaticamente il numero di revisione ogni volta che si crea un eseguibile per un progetto.
Si possono introdurre anche altre informazioni relative alla versione, tra cui dei commenti sull'eseguibile, il nome della società che lo ha creato, la descrizione del file, il copyright, i marchi commerciali e il nome specifico del prodotto.

Queste informazioni incorporata e numero di versione, copyright, e così via sono quello che gli utenti del programma vedranno quando faranno clic con il tasto destro del mouse sull'eseguibile compilato e selezioneranno la scheda Versione *del foglio di proprietà del programma.*

La scheda *Compile* della finestra di dialogo *Project Properties,* mostrata nella Figura 2.28, serve a determinare se l'eseguibile deve essere compilato in P-codice o come codice nativo autonomo. (Per ulteriori informazioni sulle conseguenze di questa decisione, si veda il Capitolo 16.)

Figura 2.27
Lafinestra di dialogo "Project Properties" serve a impostare le informazioni di versione, di argomenti della riga di comando, e gli argomenti per la compilazione condizionale.

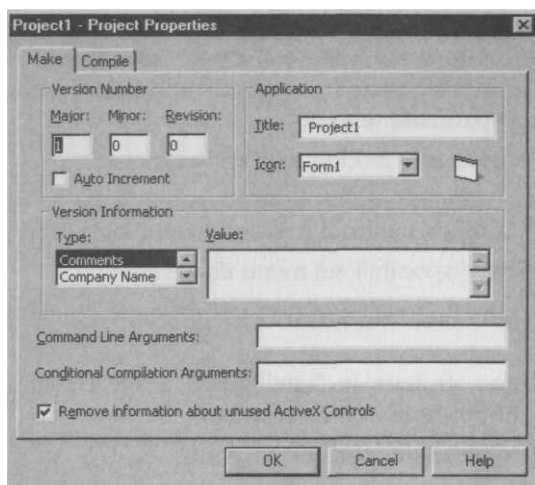
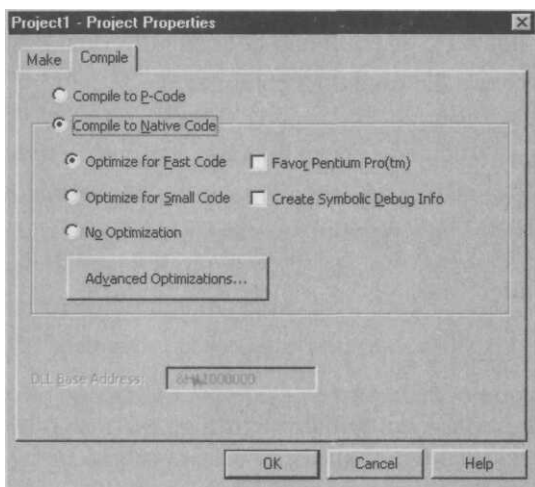
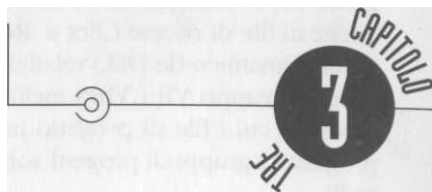



Figura 2.28
La scheda Compile della finestra di dialogo Project Properties serve a impostare la compilazione a P-codice o ad autonoma.

Riepilogo

Questo capitolo ha presentato una panoramica degli strumenti disponibili nell'ambiente di sviluppo integrato di Visual Basic che aiutano a creare applicazioni VB complete. Sono state poste in risalto le nuove caratteristiche della versione 6.

- Abbiamo visto l'IDE di Visual Basic: la barra del titolo, la barra dei menu, la toolbar o barra degli strumenti, la Toolbox o casella degli strumenti, il Project Explorer, la finestra *Properties,* la finestra *Form Layout* e la finestra di codice.
- Abbiamo visto come aprire progetti o gruppi VB esistenti, e come iniziarne di nuovi.

- Abbiamo compreso i diversi tipi di file sorgente Visual Basic, e come si incastrano insieme.
- Abbiamo visto come usare i template (modelli) di modulo e di progetto per riutilizzare il codice e risparmiare tempo.
- Abbiamo visto come aggiungere moduli a un'applicazione.
- Abbiamo visto come aggiungere controlli ai forni mediante la Toolbox e come aggiungere alla Toolbox controlli personalizzati e oggetti inseribili.
- Abbiamo visto le opzioni disponibili sui menu *Edit* e *Format.*
- Abbiamo visto come usare il Project Explorer per navigare rapidamente nei progetti.
- Abbiamo scoperto come sfruttare la "intelligenza artificiale" incorporata nell'editor di codice per produrre codice accurato.
- Abbiamo visto come creare menu per i propri form.
- Abbiamo compreso come si impostano le opzioni relative a *Environment, Project* e *Editor.*
- Abbiamo visto come utilizzare gli strumenti di debug di Visual Basic.
- Abbiamo visto come creare file eseguibili compilati.

EVENTI E OGGETTI

- Lavorare con i file sorgente di Visual Basic
- I form di Visual Basic
- Programmazione guidata da eventi
- Proprietà e metodi in Visual Basic
- Ordine di scatto per gli eventi dei form
- Uso della funzione MsgBox
- Aggiunta di codice agli eventi di form e di controllo
- Concetti basilari della programmazione orientata agli oggetti (OOP)
- Una tecnica efficace per rendere modulari e incapsulare i form e le finestre di dialogo di Visual Basic
- Che cosa sono i moduli di classe
- Creare, far scattare e gestire gli eventi personalizzati

Questo capitolo spiega la meccanica di base della moderna programmazione guidata da eventi *(event-driven programming)* nel contesto di Visual Basic. Vengono spiegati gli eventi, le proprietà e i metodi. Viene trattato il ciclo di vita di un form dall'avvio alla terminazione. Poi vengono presentati i concetti fondamentali della programmazione orientata agli oggetti *(object-oriented programming)*. Viene mostrato come incapsulare una finestra di dialogo VB e vengono affrontati i moduli di classe. Infine, viene fornita una ricetta semplice per creare e far scattare eventi personalizzati.

Lavorare con i file sorgente Visual Basic

Capitolo 2 ha spiegato i numerosi tipi di file che possono costituire un progetto Visual Basic (la Tabella 2.1 comprende ben 25 tipi di file). Riguardo a molti di questi tipi di file, i programmatori che operano con i file sorgente Visual Basic non hanno realmente bisogno di conoscere niente di più del fatto che esistono. Tuttavia, sarà opportuno conoscere meglio alcuni dei file sorgente Visual Basic con cui si lavora.

Il codice sorgente di un programma Visual Basic che possa venire compilato -"vive" in un numero arbitrario di file dei tipi più diversi. Un piccolo progetto potrebbe essere costituito da un solo modulo di forni Visual Basic (un file .Frm). Se il progetto fosse leggermente più complesso, potrebbe comprendere anche dei moduli di codice VB (i file .Bas) e dei moduli di classe (i file .Cls). Questi file di codice sorgente sono collegati in un file di progetto Visual Basic (.Vbp), che fa riferimento anche ai file di risorse (.Frx e .Res), ai controlli (i file .Ocx), e alle librerie a collegamento dinamico (le DLL) relativi al progetto.
I file di gruppo VB (.Vbg) includono riferimenti ai file di progetto VB nello stesso modo in cui i file di progetto includono riferimenti ai moduli che fanno parte del progetto. I gruppi di progetti sono particolarmente utili quando si collaudano i controlli.
Il codice sorgente di un controllo è contenuto in un modulo di controllo (.Ctl). Le pagine di proprietà che fanno parte del controllo hanno bisogno dei file sorgente .Pag. La creazione dei propri controlli è trattata approfonditamente nella Parte VI.
Ogni progetto sorgente di Visual Basic, che contiene il codice e l'informazione simbolica su cui è basato un eseguibile finale, deve contenere un file di progetto .Vbp. (Chi programma in Visual Basic da un po' di tempo, si ricorderà che nella versione 3 e nelle precedenti questo file era chiamato file .Mak *anziché* file .Vbp.) Un file di progetto viene creato quando si inizia un nuovo progetto scegliendo *New Project* dal menu *File,* e poi si salva il progetto, mentre per aprire un progetto sorgente Visual Basic esistente si deve selezionare il suo file .Vbp. Nel Capitolo 19 approfondiremo l'anatomia dei file .Vbp.
Come minimo, oltre a un file .Vbp, un progetto sorgente Visual Basic deve contenere o un file .Bas (modulo di codice) o un file .Frm (form). Come abbiamo visto nel Capitolo 2, si possono aggiungere file .Bas o .Frm a un progetto mediante il menu *Project* dell'IDE, come mostrato nella Figura 3.1.

Che cos'è un form Visual Basic?

Unform Visual Basic è una finestra creata nell'IDE di Visual Basic con gli strumenti e le tecniche discusse nel Capitolo 2. I form sono come le "finestre" che si è abituati a usare nelle normali applicazioni per Windows. Comunque, i form VB sono dotati di un'impalcatura di eventi e proprietà che facilita grandemente le operazioni di programmazione. Questa "impalcatura" è già incorporata quando si apre un nuovo form in VB. Più avanti in questo Capitolo verranno trattati gli eventi e le proprietà dei form.

Una volta che ci sono più di un modulo o di un form in un progetto, sarà necessario dire a Visual Basic dove iniziare l'esecuzione del codice quando carica il progetto. Qualunque form può venire indicato come il form di partenza *(startupform),* nel qual caso il codice di evento di quel form verrà eseguito nell'ordine trattato più avanti in questo capitolo. L'altra possibilità è quella di avviare l'esecuzione del codice da una procedura chiamata *Mairi,* che si deve prima aggiungere a un modulo di codice .Bas.

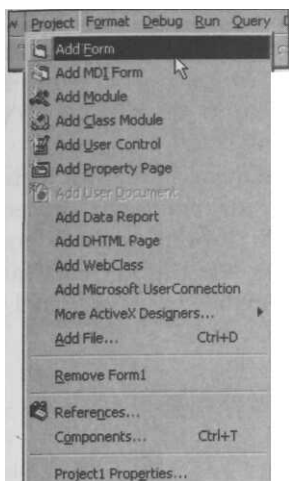
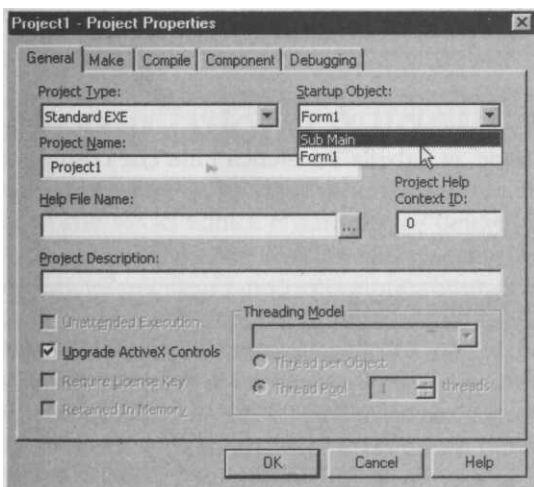
Figura 3.1
Con il menu Project si possono aggiungere un form o un altro modulo a un progetto sorgente Visual Basic.

Per scegliere il proprio form, o *Sub Main,* di partenza, bisogna aprire la finestra di dialogo *Properties* del progetto. Si può accedere alla finestra di dialogo *Properties* di un progetto dal menu *Project,* oppure facendo clic con il tasto destro del mouse sul progetto nel Project Explorer. Si seleziona la scheda *General* e si sceglie il form di partenza dalle possibilità disponibili nella casella di riepilogo a discesa, come mostrato in Figura 3.2.

Figura 3.2
La procedura Sub Main viene selezionata come oggetto di partenza.

C'è anche bisogno di sapere come aggiungere una procedura a un modulo di codice o a un form. In particolare, non si può iniziare un progetto da una *Sub Main* senza prima aver creato la *Sub Main* aggiungendola a un modulo. Inoltre, è difficile immaginare un programma VB di qualunque complessità che non contenga numerose procedure. Tratteremo l'organizzazione appropriata di queste procedure più avanti in questo capitolo e per tutta la Parte III di questo libro. Chi è interessato a questo argomento, potrebbe voler dare un'attenta occhiata al Capitolo 14.

Per aggiungere una procedura a un form o a un modulo, bisogna prima aprire il-forni o il modulo in *visualizzazione codice (code view)* nella finestra di codice, come illustrato nel Capitolo 2. Poi, si sceglie *AddProcedure* dal menu *Tools.* Si noti che, se nell'IDE non ci sono moduli o form aperti in visualizzazione codice, *Add Procedure* è in grigio e non disponibile nel menu *Tools.*
Comparirà la finestra di dialogo *Add Procedure* (Figura 3.3). Poi si dà un nome alla procedura; si decide se è una subroutine, una funzione o una procedura di proprietà e si seleziona l'ambito d'azione *(scoping).* Si possono anche impostare tutte le variabili locali di una procedura come statiche usando l'appropriata casella di controllo. Le ramificazioni di queste scelte verranno trattate in dettaglio nel Capitolo 4 e nella Parte III.

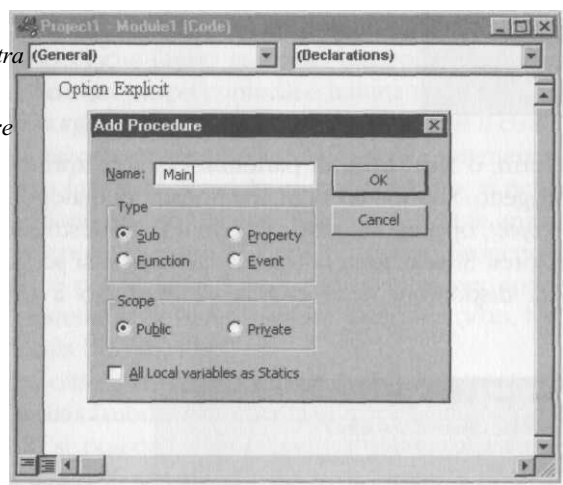
Figura 3.3 *Uso della finestra dialogo Add Procedure per aggiungere la SubMain a Modulel.*

Per riassumere questa sezione, la Tabella 3-1 elenca i file che costituiscono un progetto sorgente Visual Basic.

Tabella 3.1 *Tipi di file sorgente Visual Basic.*

File	Estensione	Scopo
File di gruppo	.Vbg	Collega un gruppo di progetti VB; particolarmente utile per eseguire dei controlli nell'ambiente di progettazione senza doverli prima compilare.
File di progetto	.Vbp	Collega tutti i file di un progetto sorgente VB. Un file di progetto è obbligatorio. In precedenti versioni di VB, era designato come file .Mak.
File di form	.Frm	Contiene il codice degli eventi e le informazioni sulle proprietà riferite a un form VB.
File di form MDI	.Frm	Contiene il codice degli eventi e le informazioni sulle proprietà riferite a un form MDI *(multiple document interface).*
Modulo di codice	.Bas	Contiene un gruppo di procedure, nel senso di subroutine e funzioni.

File	Estensione	Scopo
Modulo di classe	.Cls	File di codice sorgente di classe Visual Basic 6.
Controllo d'utente	.Ctl	Modulo sorgente su cui è basato un controllo ActiveX.
Pagina di proprietà	.Pag	Contiene il codice degli eventi e le informazioni sulle proprietà riferite a una pagina di proprietà di un controllo.
Documento d'utente	.Dob	Modulo sorgente su cui è basato un documento ActiveX.
File di risorse di form	.Frx	Contiene informazioni binarie riferite a un form, come icone e altre informazioni visuali che sono state aggiunte a un form.
File di risorse esterne	.Res	Contiene risorse esterne come stringhe di testo e bitmap in strutture standard per Windows.
Controllo ActiveX	.Ocx	Un controllo ActiveX compilato che può essere necessario per un progetto VB. I controlli ActiveX possono venire scritti in Visual Basic o in un altro linguaggio come Visual C++.
Libreria a collegamento dinamico	.Dll	Una libreria esterna compilata che può essere necessaria per un progetto VB. Gli eseguibili Visual Basic compilati come P-codice hanno bisogno del modulo DLL di supporto a tempo di esecuzione di Visual Basic. Con gli eseguibili VB6, questo file è Vbrun500.Dll. Un progetto può avere bisogno di file .Dll aggiuntivi.

La programmazione guidata da eventi

Gli *eventi* sono procedure (subroutine) che *scattano* in risposta a condizioni specifiche, nel senso che se del codice è stato posto "dentro" l'evento, tale codice viene eseguito. Tutti i programmi per Windows sono costruiti intorno agli eventi, e le applicazioni Visual Basic non fanno eccezione. Comunque, VB rende molto facile programmare gli eventi e la risposta a questi eventi. Nella tradizionale programmazione per Windows priva di un'impalcatura di eventi precostituita, rispondere da programma all'attivazione di eventi può essere una questione davvero tediosa.
Che cos'è un evento? Il concetto è intuitivamente ovvio: un evento scatta (parte) quando qualcosa (l'evento) accade. Conosciamo già bene le azioni degli utenti che fanno scattare molti eventi comuni. Alcuni esempi sono:

- Fare clic con il mouse, che fa scattare l'evento Click
- Usare la tastiera per introdurre del testo, che fa scattare gli eventi KeyDown, KeyPress e KeyUp
- Chiudere una finestra, che fa scattare l'evento QueryUnload e poi l'evento UnLoad

Come mostrano questi esempi, un evento può, semplicemente scattando, far scattare altri eventi. Si dovrebbe anche sapere che molti eventi scattano senza l'intervento dell'utente. Un evento può far scattare una cascata di altri eventi. In molte circostanze, anche il codice, il software, i sistemi operativi e l'hardware possono far scattare eventi.
La programmazione guidata da eventi è stata presentata come un concetto radicalmente diverso dalla tradizionale programmazione lineare. L'idea è che in tempi passati, prima della conquista delle interfacce utente grafiche (Graphical User Interface, abbreviato in GUI), la maggior parte del codice di programma andava in linea retta. Eventualmente, il codice permetteva la diramazione a seconda dell'input dell'utente, ma altrimenti ogni cosa era precisamente unidirezionale.
In sostanza, ci sono ben tre ragioni per cui in un ambiente guidato da eventi non si può seguire questa impostazione lineare a binario unico:

- L'esecuzione del programma in generale attende che scattino degli eventi.
- L'ordine dell'esecuzione degli eventi è complesso e non sempre interamente prevedibile.
- Possono scattare moltissimi possibili eventi.

Ciò significa che il codice di programma guidato da eventi deve essere predisposto a rispondere agli eventi non appena scattano. È una semplificazione eccessiva ma concettualmente valida, ma si può pensare che il programma attenda le azioni dell'utente invece di fare qualcosa per conto suo.
Nella programmazione tradizionale per Windows, l'impalcatura interna è qualcosa di simile a una gigantesca istruzione *case* dove ogni opzione di selezione risponde a un possibile messaggio Windows. Per esempio, selezionando una voce di menu si invia il messaggio `WM_COMMAND`. L'istruzione *case* avrebbe una diramazione per la ricezione dei messaggi `WM_COMMAND` con una sottodiramazione per l'effettiva voce di menu selezionata. Nel Capitolo 11 verrà approfondita la relazione tra Visual Basic e il sistema di messaggistica di Windows.
Ovviamente, questo può essere un approccio poco maneggevole. Può produrre programmi impenetrabili e difficili da correggere. Nel bene o nel male, Visual Basic è un ambiente di programmazione di alto livello, per lo sviluppo rapido di applicazioni (Rapid Application Development, abbreviato in RAD). Ciò isola il programmatore dal dover comprendere i dettagli del sistema di messaggistica di Windows.
Quello che fa Visual Basic è presentare eventi modello con i suoi form e controlli. Inoltre, i form e i controlli appena installati hanno già un minimo di funzionalità, perciò non ci si deve preoccupare di molti tipi di dettagli. Per esempio, quando lo si aggiunge a un progetto, un nuovo form può venire ridimensionato e chiuso senza ulteriore lavoro per il programmatore, ed è già dotato dei pulsanti per ridurlo a icona e per ingrandirlo a pieno schermo, nonché del menu di sistema.

Se si basa un nuovo progetto sull'Application Wizard di VB, gran pane della funzionalità iniziale dell'interfaccia utente viene costruita dal wizard, in base alle opzioni che vengono selezionate. Per avere informazioni sull'Application Wizard migliorato di VB6, consultare il Capitolo 6.

Un form standard è fornito di molti eventi predefiniti. Tuttavia, lo scatto di questi eventi non ha quasi alcun effetto, a meno che venga aggiunto loro del codice. Se si apre il codice associato a uno di questi eventi modello, si troveranno un inizio e una fine di procedura, e niente nel mezzo. Per esempio, ecco il codice modello standard dell'evento Form_Click:

```
PrivateSubForm_Click()
End Sub
```

Di per sé, questo gestore di evento non fa niente, anche se scatta quando l'utente fa clic sul form quando il progetto è in esecuzione. Facciamogli fare qualcosa per dimostrare ulteriormente il concetto dell'impalcatura di gestione di eventi.

Utilizzo della funzione MsgBox quando scatta un evento

La funzione MsgBox è molto facile da usare ed è un modo sorprendentemente flessibile di passare messaggi agli utenti di un programma. Essa permette anche di ottenere semplici risposte a qualunque domanda si riveli necessario porre ai propri utenti.
Si può anche usare questa funzione come strumento di debug. È facile collocare una semplice funzione MsgBox in un gestore di eventi per vedere se l'evento è effettivamente scattato. Inoltre, se non c'è bisogno di ottenere un valore di risposta, la si può codificare come istruzione invece che come funzione. Per esempio, si potrebbe aggiungere

```
MsgBox "Sono scattato!"
```

all'evento Form_Paint, se si volesse verificare che l'evento *Paint* fosse scattato.
Per avere informazioni complete sulla funzione MsgBox, la cosa migliore è cercarla nella guida in linea di VB. Si può anche usare efficacemente l'Object Browser per trovare tutti i possibili valori dell'icona e delle costanti rese che si possono usare nella funzione MsgBox.
Si può trovare la funzione MsgBox selezionando la libreria VBA nell'Object Browser. Successivamente, selezionare l'oggetto VBA.Interaction. Tra i membri di questo oggetto c'è MsgBox.

VBA (Visual Basic far Applications) è un sottoinsieme del linguaggio Visual Basic che è ampiamente utilizzato come linguaggio di macro all'interno di altri prodotti Microsoft, come la suite Office.

Siccome le costanti di MsgBox sono predefinite come parte di VBA, si possono usare le parole elencate equivalenti senza doverle dichiarare. Per esempio, vbCancel è predefinito come uguale a 2. È una migliore pratica di programmazione usare il termine costante descrittivo invece del meno leggibile equivalente numerico. L'unica ragione per usare i valori numerici è mantenere la compatibilita di codice all'indietro con VB3 e le versioni precedenti. Le costanti di MsgBox si possono sommare; per esempio, si potrebbero volere i pulsanti *Yes, No* e *Cancel* e un'icona esclamativa.

Effettivamente, con le nuove caratteristiche di "intelligenza artificiale" di VB6, non c'è in realtà bisogno di cercare niente. Basta digitare **MsgBox** *nella finestra di codice, premere la barra spaziatrice, e VB fa saltar fuori la sintassi completa dell'istruzione* MsgBox. *Quando ci si sposta ai parametri dell'istruzione* MsgBox *che si attendono dette costanti, VB produce una casella di riepilogo a discesa delle varie possibilità. Basta fare doppio clic su una costante della casella, e VB la inserisce nell'istruzione.*

Aggiunta di codice a un evento Click di un form

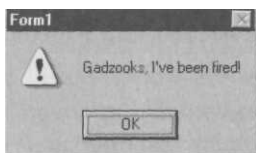
Per aggiungere un'istruzione MsgBox all'evento *Click* di un form, bisogna aprire la finestra di codice quando quel form è attivo. Mediante la casella di riepilogo a discesa *Object* nell'angolo superiore sinistro della finestra di codice si può selezionare il form. Una volta che il form è selezionato, si usa la casella di riepilogo a discesa *Procedure* nel lato superiore destro della finestra di codice per selezionare l'evento *Click.* Quando si seleziona l'evento *Click,* VB automaticamente crea il modello del codice di gestione:

```
Private Sub Form_Click()

End Sub
```

Si aggiunge quindi l'istruzione MsgBox, o dell'altro codice che si desidera, entro la procedura di gestione:

```
Private Sub Form_Click()
   MsgBox "Gadzooks, I've been fired!", vbExclamation, Me.Caption
End Sub
```

A questo punto, quando l'utente esegue questo programma d'esempio e fa clic sul form, viene visualizzata una casella di messaggio, come si può vedere nella Figura 3.4. La parola chiave Me.Caption fa sì che sia visualizzata la didascalia del form corrente.

Figura 3.4
È scattato un evento clic ed è comparsa una finestra di messaggio.

Questo capitolo, e i capitoli seguenti, dimostrano molte altre tecniche di programmazione guidata da eventi. Nel frattempo, possiamo concludere questa panoramica della programmazione guidata da eventi con un commento secondo il mio personale punto di vista.
Ho incominciato a programmare fin dai giorni in cui si passava un mazzo di schede perforate attraverso un lettore per mandarle al bestione della stanza accanto. (Qualcun altro si ricorda dell'era dei dinosauri?) Questi erano sicuramente programmi

lineari non guidati da eventi. Non solo non era coinvolta alcuna risposta dell'utente, ma ci volevano ore (o giorni) per ottenere l'output, e poteva darsi che quell'output consistesse

Questo modello di programmazione sembrerebbe molto diverso dagli standard attuali delle GUI, in cui un programmatore prepara sullo schermo l'interfaccia utente e poi, normalmente, attende l'azione dell'utente. Come nota collaterale qui, la schiacciante popolarità di Windows e di altre GUI fra gli utenti dimostra chiaramente la superiorità di questo stile dal punto di vista dell'utente, sebbene come contropartita abbia una maggiore complessità di programmazione.
Quello che voglio dire qui è che in realtà non credo che la programmazione guidata da eventi sia, in astratta teoria, qualitativamente diversa dal vecchio stile di programmazione. Piuttosto, è più complessa ma non meno lineare in un senso logico. Il codice semplicemente ha bisogno di anticipare una risposta a molte più possibili cose (leggasi "eventi"). Credo che sia d'aiuto tenere a mente questo, quando si progettano programmi guidati da eventi: un programma guidato da eventi è meramente un programma lineare molto più complesso. Non anticipare ogni possibile evento in questo contesto è un peccato tanto mortale come permettere un input non lecito dell'utente in un programma non guidato da eventi. Fortunatamente, come vedremo, l'impalcatura di eventi di Visual Basic fornisce un modo facile e diretto per gestire gli eventi.

Proprietà e metodi in Visual Basic

Oltre agli eventi, la maggior parte degli oggetti Visual Basic, come i form e i controlli, è fornita di *proprietà* e di *metodi.* I metodi e le proprietà sono concettualmente fondamentali per il modo in cui funzionano gli oggetti; sono i blocchi da costruzione basilari per operare con VB.

Le proprietà

Una proprietà è un'impostazione che descrive qualcosa riguardo un oggetto come, per esempio, un form. A seconda della proprietà, la si può impostare in fase di progettazione mediante la finestra *Properties* (vedere il Capitolo 2 per ulteriori informazioni), o in fase di esecuzione con istruzioni nel codice. Per avere un elenco completo con spiegazione delle proprietà e dei metodi, oltre che degli eventi, dei form, cercare "Form Object" nella guida in linea di Visual Basic.
Ecco due esempi delle proprietà dei form Visual Basic:

MinButton. Questa proprietà può venire impostata a *True* o a *False.* A seconda dell'impostazione, il form ha (o non ha) un pulsante di riduzione a icona.

BackColor. L'impostazione di questa proprietà a un valore espresso come un RGB esadecimale o a una costante modifica il colore di sfondo del form. Si possono cercare le costanti usando l'Object Browser nella libreria VBRUN sotto "ColorConstants" e "SystemColorConstants".

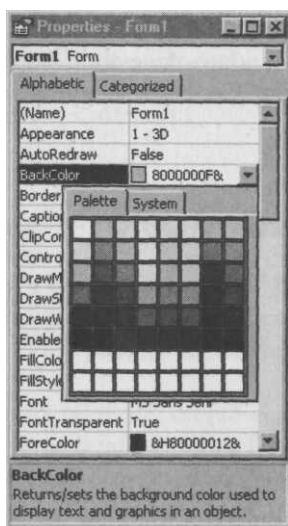
Per esempio, per aggiungere una riga di codice all'evento doppio clic di un form in fase di esecuzione, basta modificare il gestore di eventi come segue:

```
Private Sub Form_DblClick()
   BackColor = vbRed
End Sub
```

Tra l'altro, l'equivalente esadecimale è:

```
BackColor = &HFF&.
```

Si può impostare la stessa modifica di colore, sebbene non in risposta allo scatto di un evento, usando la finestra proprietà in fase di progettazione. Se si fa clic sulla proprietà *BackColor,* si otterrà una tavolozza di colori come quella mostrata nella Figura 3.5. L'aspetto di questa tavolozza dipende dalle impostazioni del dispositivo grafico utilizzato e dalle impostazioni selezionate nel Pannello di controllo di Windows.

Figura 3.5
La proprietà BackColor di un form può venire impostata mediante la finestra Properties.

I metodi

I *metodi,* rispetto alle proprietà e agli eventi, sono procedure che agiscono su un oggetto. Internamente, i metodi sono scritti come funzioni. In generale, possono solamente venire eseguiti in fase di esecuzione, non in fase di progettazione. Alcuni esempi dei metodi dei form sono il metodo Move, che sposta un form nello spazio bidimensionale dello schermo, e il metodo ZOrder, che posiziona il form di fronte o dietro altre finestre.
I metodi vengono *invocati* scrivendo il nome dell'oggetto il cui metodo viene chiamato, seguito dall'operatore punto (.), seguito dal nome del metodo. Come ogni routine, i metodi possono prendere argomenti. Per esempio:

```
Form1.Zorder 0
```

Per fare un esempio, aggiungeremo una chiamata al metodo Form.Move per centrare il form quando viene caricato. Il codice necessario per farlo, collocato nell'evento Form_Load, è mostrato nel Listato 3.1.

Listato 3.1 *Centrare unform sullo schermo.*

```
Private Sub Form_Load()
   Me.Move (Screen.Width - Me.Width) \ 2,
     (Screen.Height - Me.Height) \ 2
End Sub
```

Oltre al metodo Form.Move, questo frammento di codice usa alcune semplici tecniche VB che si dovrebbe conoscere per centrare qualunque form sullo schermo. La parola chiave Me in VB si riferisce all'esemplare di form (o di modulo di classe) in esecuzione in quel momento. La parola chiave Screen si riferisce a un oggetto VB che in fase di esecuzione controlla il posizionamento dei form sullo schermo (come anche l'aspetto del cursore). .Width e .Height sono due proprietà dell'oggetto Screen. Per avere un elenco completo delle sue proprietà disponibili, lo si può cercare nella guida in linea. Infine, in VB6, uno spazio seguito da un carattere di sottolineatura (_) viene usato per continuare una riga (logica) di codice su una riga (fisica) successiva. Lo scopo principale di questa convenzione è semplicemente accrescere la leggibilità del codice.

Per quanto questa procedura per centrare unform sia istruttiva, poichépuò risultare necessario spostare unform in vari modi, non è realmente indispensabileper centrare un form sullo schermo quando viene aperto la prima volta. Perfarlo, basta impostare a 2-CenterScreen *laproprietà* StartUpPosition *delform nellafinestra* Properties.

Ai form si possono aggiungere facilmente proprietà e metodi. Questo argomento verrà trattato nel Capitolo 13.

Ordinamento di scatto degli eventi

Per controllare con successo l'aspetto e il comportamento in fase di esecuzione dei form (e anche dei controlli), si deve comprendere l'ordine secondo cui gli eventi scattano. Le questioni sull'ordine di scatto degli eventi vengono in generale risolte a beneficio del programmatore nel punto in cui dovrebbe venire collocata una data porzione di codice di risposta all'evento. Si possono suddividere gli eventi dei form nei seguenti gruppi:

- Avvio *(start up)*
- Risposta alle azioni dell'utente
- Collegamento (*linking)*
- Arresto *(shut down)*

Tratteremo gli eventi di collegamento quando passeremo all'Automazione OLE nella Parte V del libro. Per adesso guardiamo i gruppi di eventi della nascita (avvio), della vita (risposta), e della morte (arresto): che cosa fanno e quando scattano.
Va notato che un dato controllo ha un insieme di eventi diverso da quello di un form. Ciò significa che mentre i concetti di evento trattati in questa sezione spesso si applicano ai controlli come anche ai form, la corrispondenza non è sempre completa. Comunque, questa sezione dovrebbe dare un'idea del tipo di comportamento che ci si può, in generale, aspettare dagli eventi, indipendentemente da quale oggetto li faccia scattare. Inoltre, quando creerete i vostri controlli, divrete distinguere chiaramente fra il ricevere e il far scattare un evento da parte di un controllo. Questo argomento verrà trattato nel Capitolo 24.
È anche importante comprendere che un evento spesso fa scattare automaticamente un altro evento, innescando un effetto a cascata. Per esempio, un evento KeyPress non può scattare senza far scattare anche gli eventi KeyUp e KeyDown. Il segreto per operare con questo tipo di situazioni è comprendere chiaramente ciò che fa scattare ogni evento della successione; il pericolo che si corre nella codifica è far partire una catena senza fine di chiamate di evento ricorsive circolari.

Gli eventi di avvio dei form

Si possono mettere in moto gli eventi di nascita (o avvio) di un form o specificando tale form come form di avvio del progetto, come descritto prima in questo capitolo, o invocando il metodo Form.Show, per esempio:

```
Form1.Show
```

Il metodo Show (mostra) carica e visualizza un form; sono due azioni distinte che fanno parte entrambe del processo di nascita. Anche quando si esegue il form di avvio, prima il form viene caricato e poi viene mostrato. Il metodo Show (per un form non MDI) può venire invocato come *non a scelta obbligatoria (modeless)* o come *a scelta obbligatoria (modal).* A scelta obbligatoria o *modal* significa che nessun codice successivo viene eseguito finché il form non viene nascosto o scaricato. Quando un form a scelta obbligatoria viene visualizzato, l'utente non può effettuare alcun input (con la tastiera o con il mouse) se non verso gli oggetti presenti sul form a scelta obbligatoria. I form non a scelta obbligatoria, al contrario, non monopolizzano necessariamente l'azione!
L'obbligatorietà della scelta viene specificata tramite un parametro di stile che segue la chiamata del metodo Show. Se lo stile non è specificato o è O, il form è non a scelta obbligatoria (come appena descritto); se è 1, il form è a scelta obbligatoria. Per esempio:

```
Form1.Show 1
```

avvia il Form1 a scelta obbligatoria.

Quando un form si avvia, scattano in sequenza i seguenti eventi:

1. **Initialize**- Serve a inizializzare i dati utilizzati da un esemplare del form in modo che saranno già pronti nel form, e visibili all'utente, quando il form viene caricato.
2. **Load.** Serve per eseguire ulteriore codice di inizializzazione. Inizia il caricamento del form in memoria e lo visualizza. Bisogna stare attenti a non effettuare chiamate potenzialmente circolari (ricorsive) ad altri metodi di form come Activate, Resize, Paint e GotFocus entro questo metodo.
3. **Resize.** Avviene quando un form viene visualizzato per la prima volta. Scatta anche ogni volta che il form viene ridimensionato (perciò, come anche l'evento Paint, potrebbe essere considerato un evento di vita oltre che un evento di nascita!).
4. **Paint.** Avviene quando una parte di un form o tutto il form viene scoperto dopo che è stato spostato o allargato o dopo che una finestra che lo stava coprendo è stata spostata. L'evento Paint viene invocato anche quando il metodo Refresh viene chiamato in fase di esecuzione. Se la proprietà AutoRedraw del form è impostata a True, il ridisegno è automatico, perciò probabilmente in tal caso non c'è alcun motivo di aggiungere del codice all'evento Paint.

Gli eventi di risposta dell'utente dei form

Per la vera natura della progettazione di programmi guidati da eventi, non c'è quasi mai modo di prevedere esattamente quali azioni dell'utente in fase di esecuzione faranno scattare eventi specifici. In altre parole, questi eventi non possono venire ordinati semplicemente come la sequenza di avvio del form, tuttavia, fino a un certo punto, possono essere raggnippati. (Nelle sezioni sulla programmazione del trascinamento del Capitolo 21 tratteremo molto più approfonditamente come funzionano gli eventi relativi al mouse.)
Molti eventi di risposta dell'utente restituiscono parametri che contengono importanti informazioni sull'evento che è scattato; per esempio, la posizione del mouse quando è stato fatto un clic, o quale tasto è stato premuto. Discuteremo più dettagliatamente questi eventi nei luoghi appropriati.

Gli eventi del mouse

Gli eventi del mouse a livello del form sono, in ordine alfabetico:

- **Click.** Scatta quando viene fatto clic con un pulsante del mouse.
- **DblClick.** Scatta quando viene fatto clic due volte con un pulsante del mouse entro i limiti di tempo impostati nel Pannello di controllo. Notare che facendo scattare questo evento in realtà si fa scattare una sequenza di eventi: MouseDown, MouseUp, Click, DblClick, e MouseUp. Mentre l'espressione inglese per questo evento è "double-click", il nome interno dell'evento è, come appare in questo elenco, "DblClick".

- **DragDrop.** Avviene quando viene completata un'operazione di trascinamento.
- **DragOver.** Scatta continuamente quando un'operazione di trascinamento è in corso. Questo evento serve a monitorare il puntatore dei mouse man mano che entra, esce, o rimane direttamente su un valido oggetto bersaglio, come un form. La posizione del puntatore del mouse determina l'oggetto bersaglio che riceve questo evento.
- **MouseDown.** Scatta quando viene premuto un pulsante del mouse.
- **MouseMove.** Scatta quando il mouse viene spostato.
- **MouseUp.** Scatta quando viene rilasciato un pulsante del mouse.

Gli eventi della tastiera

Ecco gli eventi della tastiera a livello del form:

- **KeyDown.** Scatta quando viene premuto un tasto.
- **KeyPress.** Scatta quando un tasto viene premuto e poi rilasciato.
- **KeyUp.** Scatta quando viene rilasciato un tasto.

Altri eventi

Gli altri eventi comprendono gli eventi Paint e Resize già trattati nella sezione "Gli eventi di avvio dei form". Inoltre, bisogna conoscere l'esistenza anche dei seguenti eventi:

- **Adivate.** Scatta quando un form diventa la finestra attiva. Avviene prima che scatti l'evento GotFocus.
- **DeActivate.** Scatta quando un form cessa di essere la finestra attiva. Scatta prima di LostFocus.
- **GotFocus.** Avviene quando un form (o un controllo) riceve lo stato attivo (focus); cioè, diventa la sola finestra che possa ricevere input da tastiera e da mouse in quel momento. Normalmente, si può stabilire quando un form ottiene lo stato attivo perché la sua barra del titolo cambia colore. Quando un oggetto come un pulsante di comando ottiene lo stato attivo, l'aspetto della sua didascalia cambia. Per esempio, intorno al nome del pulsante di comando compare una linea tratteggiata. Affinchè un oggetto possa ricevere lo stato attivo, Le sue proprietà Enabled e Visible devono essere impostate a True.
- **LostFocus.** Scatta quando un form (o un controllo) perde lo stato attivo.

Gli eventi di chiusura dei form

Ecco gli eventi che scattano in un form quando si chiude, in ordine di scatto:

1. **QueryUnload.** Fatto scattare dall'evento Unload di un form, prima che sia eseguito il codice dell'evento UnLoad. QueryUnload dà l'opportunità di

bloccare lo scaricamento del form dalla memoria; per esempio, quando l'utente ha modificato dei valori nel form senza aver salvato le modifiche. Il form non verrà scaricato se la variabile Cancel della procedura di evento QueryUnload viene impostata a True. È pratica comune impostare questa variabile a True, quando l'utente risponde a una richiesta del tipo "Salvare le modifiche? Sì, No, Annulla" optando per annullare. Si può usare facilmente la funzione MsgBox per dare all'utente l'opportunità di annullare uno scaricamento; si veda la prossima sezione per avere un esempio di come farlo. È anche utile che un parametro UnloadMode reso dalla procedura di evento dica la fonte dell'evento QueryUnload che è scattato, rispondendo alle seguenti domande: È stato l'utente che ha fatto clic sul pulsante *Chiudi?* È stata l'esecuzione del codice a provocare l'evento Unload? È il Task Manager di Windows che sta chiudendo l'applicazione? Questa informazione consente di intraprendere diverse azioni a seconda di che cosa stia chiudendo l'applicazione.

2. **Unload.** Scatta quando un utente chiude il form con il comando *Chiudi* sul menu di controllo o quando un metodo Unload viene eseguito nel codice. Unload fa scattare immediatamente un evento QueryUnload come appena descritto. Si può usare l'evento Unload (come anche QueryUnload) per eseguire compiti di chiusura come salvare e convalidare i dati. Come con QueryUnload, si può arrestare lo scaricamento anche impostando a True il parametro Cancel dell'evento. Comunque, solitamente è meglio eseguire questa azione nell'evento QueryUnload, perché da un evento Unload non si può impedire l'arresto di Windows stesso.
3. **Terminate.** Scatta quando tutti i riferimenti a un esemplare di un form sono stati tolti dalla memoria.

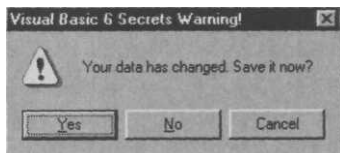
La funzione MsgBox e QueryUnLoad

Il Listato 3.2 dimostra come, nell'evento QueryUnLoad di un form, si possa utilizzare una funzione MsgBox che verifichi se l'utente vuole salvare i dati modificati o annullare l'uscita, come spiegato nella sezione precedente. La Figura 3.6 mostra l'avvertimento visualizzato dalla routine.

Listato 3.2 *Utilizzo dell'evento QueryUnLoad.*

```
Private Sub Form_QueryUnload(Cancel As Integer, _
   UnloadMode As Integer)
   Dim Response As Integer
   Response = MsgBox("Your data has changed. Save it now?", _
      vbYesNoCancel+vbExclamation,_
      "Visual Basic 6 Secrets Warning!")
   Select Case Response
      Case vbYes
         MsgBox "Save the data and unload!"
      Case vbNo
```

```
            MsgBox "Don't save the data and unload!"
        Case vbCancel
            MsgBox "Don't Unload!"
            Cancel = True
        Case Else
            MsgBox "Internal Error!"
    End Select
End Sub
```

Figura 3.6 *Si può usare la funzione MsgBox per visualizzare avvenimenti all'utente.*

Aggiunta di codice agli eventi dei form e dei controlli

Questo capitolo ha già mostrato i modi generali in cui si aggiunge codice eseguibile a una procedura di gestione di eventi. È importante, comunque, rivedere esplicitamente come raggiungere l'appropriata intelaiatura di eventi in cui collocare questo codice. Ciò permette di scegliere il modo più veloce e più facile per arrivare all'intelaiatura di gestione di eventi a cui si desidera aggiungere il proprio codice. Dopo aver raggiunto l'appropriata procedura di gestione di eventi, per inserire il codice basta digitarlo nella finestra di codice, come abbiamo visto negli esempi di questo capitolo. I pratici strumenti linguistici di VB6 aiutano a trovare la corretta sintassi in ogni situazione.

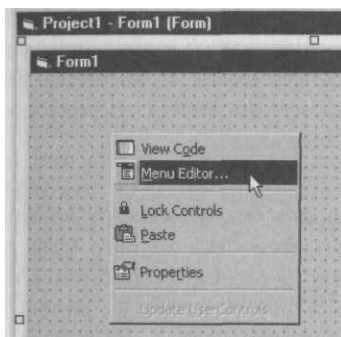
È importante rendersi conto che il codice inserito in una procedura di gestione di eventi spesso consiste semplicemente in chiamate di procedure. In tali casi, le procedure chiamate contengono l'effettivo codice eseguibile. Una ragione per progettare un programma in questo modo è che la stessa procedura può venire chiamata da molti diversi gestori di eventi, in modo da semplificare, abbreviare, e chiarificare l'architettura del programma. Una tecnica comune è di passare a una procedura chiamata da un gestore di eventi un parametro che indica quale gestore l'ha chiamata. L'esecuzione nella procedura chiamata può allora seguire una diramazione a seconda di quale procedura l'ha chiamata, come determinato dal parametro.

Ecco i tre modi per "raggiungere" un'intelaiatura di procedura di gestione di eventi:

- Assicurarsi che il Project Explorer sia aperto; se necessario, scegliere *Project Explorer* dal menu *View.* Selezionare il form a cui si desidera aggiungere del codice di evento; se si sta aggiungendo un evento a un controllo che è stato collocato su un form, selezionare il form che è il "genitore" del controllo. Fare clic sul pulsante *View Code;* alternativamente, fare clic con il tasto destro sull'icona del form e scegliere *View Code* dal menu di scelta rapida. Nella casella di riepilogo *Object,* selezionare il form o un altro oggetto (per esempio, un controllo) a cui si vuole aggiungere del

codice di evento. Successivamente, dalla casella di riepilogo *Procedure,* selezionare il gestore di eventi a cui si aggiungerà del codice. Notare che le procedure di gestione di eventi con del codice inserito appaiono in grassetto nella casella di riepilogo *Procedure,* mentre quelle che non contengono codice appaiono normali.

- Fare doppio clic sul form a cui si vuole aggiungere del codice. Fare le proprie selezioni dalla casella di riepilogo *Object* e dalla casella di riepilogo *Procedure* come appena descritto.
- Fare clic con il pulsante destro del mouse sul form. Scegliere *View Code* dal menu di scelta rapida (Figura 3.7). Effettuare le proprie selezioni dalla casella di riepilogo *Object* e dalla casella di riepilogo *Procedure* come appena descritto.

Figura 3.7
Facendo un clic con il pulsante destro del mouse su un form, viene visualizzato un menu di scelta rapida con le opzioni che riguardano il form.

Concetti fondamentali di programmazione orientata agli oggetti

Le tecniche di programmazione orientata agli oggetti *(object-oriented programming,* in sigla OOP) sono fra gli strumenti più importanti che in questo decennio si sono aggiunti al repertorio della maggior parte degli sviluppatori. I programmi che sono stati progettati con l'OOP in mente tendono a essere riutilizzabili e manutenibili con una coerenza molto maggiore. I programmi OOP sofisticati possono anche essere altamente estendibili, nel senso che potenzialmente possono simulare il cambiamento e la crescita come organismi viventi e sono adatti alPimplementazione di sistemi esperti e di costrutti di intelligenza artificiale. Lo sviluppatore professionista non può assolutamente permettersi di ignorare l'OOP.

Comunque, non esiste alcun OOP standard o condiviso, e gli strumenti OOP disponibili agli sviluppatori dipendono dagli ambienti di programmazione che usano. I linguaggi attuali offrono un'ampia gamma di strumenti per l'OOP. Lo spettro va da linguaggi OOP relativamente puri come Ada e Smalltalk, che, comunque, non sono linguaggi di sviluppo di grande diffusione, fino a linguaggi di grande diffusione come le versioni di Java, C++, e Borland Delphi per Windows, che incorporano potenti dispositivi per l'OOP.

Visual Basic è, naturalmente, il linguaggio di programmazione più venduto al mondo. La sua popolarità si fonda su molti buoni motivi: è un ambiente incredibilmente facile da usare e tuttavia potente. Come tale, Visual Basic è decisamente di grande diffusione. Ogni release successiva di Visual Basic è diventata sempre più orientata agli oggetti. Sebbene Visual Basic sia un linguaggio ibrido, VB6 fornisce strumenti molto significativi per l'OOP Lo spirito di Visual Basic è essenzialmente non dogmatico, ma con la versione 6 gli aspetti OOP dell'ambiente sono diventati di serie A.

Il supporto migliorato alle classi, la capacità di far scattare eventi *personalizzati* (o custom, cioè aggiunti dal programmatore), e la capacità di creare i propri controlli ActiveX senza lasciare l'ambiente, accrescono potentemente il vocabolario OOP disponibile agli sviluppatori VB. Per ulteriori informazioni, si veda il Capitolo 14.

Tre concetti sono centrali per le tecniche di programmazione OOP:

- **Ereditarietà.** Ereditarietà significa avere la capacità di creare un nuovo oggetto basato su un oggetto esistente. Alcuni degli eventi, delle proprietà, e dei metodi del nuovo oggetto potrebbero essere leggermente diversi da quelli del vecchio oggetto. Oppure, alcune nuove voci potrebbero venire aggiunte ai vecchi eventi, proprietà, e metodi.
- **Incapsulamento.** Questo termine indica tutta una serie di concetti. Nel senso più semplice, significa raggruppare dei dettagli implementativi di oggetti in modo che tali oggetti siano accessibili ad altre parti del programma solamente attraverso procedure di accesso predefinite. L'uso degli oggetti incapsulati favorisce appropriati livelli di "accoppiamento", ossia connessione, fra le parti di un programma. Questo è molto importante se si desidera costruire applicazioni "a prova di bomba". L'obiettivo è creare oggetti incapsulati con forte integrità interna, nota come "coesione forte", e un chiaro, diretto, e flessibile accesso fra gli oggetti, noto come "accoppiamento lasco". Se ricorderete almeno la coesione forte e l'accoppiamento lasco, probabilmente non vi troverete in difficoltà con l'incapsulamento.
- **Polimorfismo.** Secondo il dizionario, il termine "polimorfismo" si applica a un oggetto "che ha, che si verifica in, o che passa attraverso più [incarnazioni]". L'idea è che il polimorfismo abilita a implementare eventi, proprietà e metodi in oggetti derivati come sottoclassi in vari modi. Un oggetto derivato come sottoclasse è un oggetto che eredita da un oggetto concettualmente posizionato più in alto nella struttura delle classi. Si può chiamare un metodo, ma l'oggetto polimorficamente ereditato non sa come il metodo deve venire implementato finché non viene "vincolato" in fase di esecuzione.

Per comprendere meglio il concetto, si consideri come metafora il metodo "Accenditi", che potrebbe applicarsi agli oggetti Televisore e Trattore, entrambi appartenenti alla classe "Macchina". Ognuna delle macchine gestisce diversamente Pimplementazione del metodo "Accenditi", e a noi in realtà non interessa. Nel gergo dell'OOP, si potrebbe dire che la classe "Macchina" ha un metodo "Accenditi" che è stato ereditato da ognuna delle sue sottoclassi, Televisore e Trattore, e modificato da ognuna con la sua implementazione specifica.

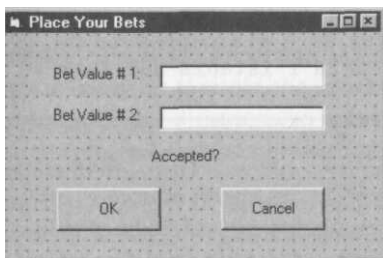
Incapsulamento delle finestre di dialogo di Visual Basic

Le tecniche corrette dell'accoppiamento e dell'incapsulamento impongono che i programmatori facciano ogni sforzo per evitare di usare variabili globali o valori fissati una volta per tutte nel codice. Evitando questi errori si rendono i programmi più facili da manutenere e gli oggetti più facili da riutilizzare. La maggior parte delle procedure interne a un oggetto forni appropriatamente incapsulato dovrebbero essere con ambito d'azione privato, e perciò inaccessibili dall'esterno del modulo. L'*ambito d'azione* o *ambito di visibilità (scope)* di una procedura o variabile indica in che misura sia accessibile agli altri moduli di un programma.
Il problema è come popolare una finestra di dialogo e come restituire valori (alla e dalla finestra) in un modo riusabile senza aggiungere procedure pubbliche all'oggetto, il che contravverrebbe le finalità dell'accoppiamento. L'ambito d'azione in VB e le sue implicazioni e conseguenze vengono trattati nel Capitolo 13.
Ecco un modo possibile di gestire tale questione:

1. Creare un modulo di codice .Bas parallelo con lo stesso nome di ogni file di form .Frm che si è creato. Usare il file .Bas per mettere e recuperare valori dal form. Siccome entrambi i moduli hanno lo stesso nome, solo le loro estensioni sono diverse, è facile trasferirli insieme ad altri progetti e riutilizzarli in seguito.
2. Usare un controllo etichetta *(Label)* invisibile per passare le informazioni avanti e indietro fra il modulo .Bas e il modulo .Frm. Per rendere invisibile un controllo si deve impostare a False la sua proprietà Visible. Il controllo etichetta non utilizza le risorse di Windows e gli si può dare un nome che lo identifica chiaramente.

Un altro aspetto evidenziato da questa tecnica è che un form VB è solamente un tipo di oggetto piuttosto speciale. Visual Basic crea automaticamente un esemplare di un oggetto form, senza che gli venga richiesto. Ma niente impedisce ai programmatori di creare e distruggere altri esemplari di form.

Vediamo come funziona questa tecnica con un esempio. Incominciamo col creare una finestra di dialogo per le scommesse come quella illustrata in modalità progettazione in Figura 3.8.

Figura 3.8
Questa finestra di dialogo per le scommesse dimostra come può funzionare il passaggio di parametri incapsulati

Per un certo non so che stilistico, assicurarsi di impostare a 3 - Fixed Dialog lo stile del bordo del forni. Chiamare txtBet1 la prima casella di testo e txtBet2 la seconda. Aggiungere un'etichetta chiamata Accepted, e controllare che la sua proprietà Visible sia impostata a False. (Questa etichetta verrà usata per passare avanti e indietro le informazioni sullo stato del form, indicando se l'utente ha accettato i valori.)

Successivamente, aggiungere un pulsante di comando chiamato cmdOK avente *OK* come didascalia e un altro pulsante chiamato cmdCancel avente *Cancel* come didascalia. Si può impostare a True la proprietà Default del pulsante *OK;* ciò fa sì che il tasto Invio attivi il suo evento Click. Impostare anche a True la proprietà Cancel del pulsante *Cancel;* ciò fa sì che il tasto Esc attivi l'evento Click di tale controllo.

Chiamare frmEncap il form e salvare il modulo del form come Encap.Frm. Aggiungere il seguente codice ai pulsanti cmdOK e cmdCancel:

```
Private Sub cmdCancel_Click()
    Accepted = vbCancel 'valore costate di VBA predefinito
    Me.Hide 'Nasconde il form ma non lo scarica
End Sub

Private Sub cmdOK_Click()
    Accepted = vbOK 'valore costate di VBA predefinito
    Me.Hide 'Nasconde il form ma non lo scarica
End Sub
```

Il passo successivo è aggiungere un nuovo modulo di codice .Bas e salvarlo come Encap.Bas. Poi aggiungere a Encap.Bas la funzione che effettivamente popolerà il form e recupererà informazioni da esso (vedere il Listato 3.3).

Listato 3.3 *Incapsulamento di unform.*

```
Private Function PlaceBets(ByVal Bet1 As String, _
ByVal Bet2 As String)As Integer
    Dim X As New frmEncap 'Crea una nuova istanza di frmEncap
    X.txtBet1 = Bet1        'Inserisce i valori iniziali
    X.txtBet2 = Bet2
    X.Show 1 'modal
    If X.Accepted = vbOK Then
        'Se è stato premuto OK, restituisce i valori correnti
        Bet1 = X.txtBet1
        Bet2 = X.txtBet2
    End If
    PlaceBets = X.Accepted 'Ritorna se è stato premuto OK
    Unload X     'Scarica dalla memoria l'istanza del form
End Function
```

E questo è il trucco! Si può verificare se questo codice effettivamente funziona aggiungendo una Sub Main al modulo .Bas e avviando il progetto da esso. Il codice nella Sub Main, in cui si possono sostituire le stringhe con valori di proprio gusto, popolerà la finestra di dialogo, e restituirà il contenuto dei campi casella di testo della finestra di dialogo se, e solo se, l'utente convalida i valori selezionando *OK.*

```
Public Sub Main()
   Dim RetVal As Integer
   Dim Bet1, Bet2 As String
   Bet1 = "$1,000"
   Bet2 = "$2,000"
   RetVal = PlaceBets(Bet1, Bet2) 'inizializza i valori
   If RetVal = vbOK Then 'L'utente ha accettato i valori
      MsgBox "Prima scommessa: " & Bet1 _
         & " ; Seconda scommessa: " & Bet2
   Else 'L'utente oggi non è un grande scommettitore!
      MsgBox "Non sono state accettate scommesse!"
   End If
End Sub
```

Che cosa sono i moduli di classe

Un *modulo di classe (class module) è* il modello per un oggetto privo di un'interfaccia utente visibile, in modo molto simile a come un modulo di forni è il modello per un oggetto che visualizza una finestra sullo schermo. Si pensi a un modulo di classe come a uno stampino per biscotti e all'oggetto di tale classe come a un biscotto. Le operazioni con i moduli di classe sono trattate in dettaglio nel Capitolo 14. La utility Class Builder di VB è trattata nel Capitolo 5.

Proprietà

Per aggiungere delle proprietà a un modulo di classe, si possono usare le procedure accoppiate Get e Let di tipo Public Property:

```
Public Property Get myProp() As Variant

End Property

Public Property Let myProp(ByVal vNewValue As Variant)

End Property
```

Per aggiungere una procedura Property *a un modulo di classe, si può usare la finestra di dialogo* Add Procedure *del menu* Tools, *assicurandosi che sia selezionato* Property *come tipo di procedura. Dopo aver aggiunto una procedura* Property *alla classe, le si possono cambiare gli attributi, visualizzabili con l'Object Browser, usando la finestra di dialogo* Procedure Attributes, *alla quale si accede sempre per mezzo del menu* Tools. *Se si vuole che una proprietà sia usata solo internamente a un modulo di classe, non è necessario usare le routine* Property Get *e* Property Let: *è sufficiente implementarla come variabile a livello di modulo.*

Per implementare la nuova proprietà aggiunta al modulo di classe, si deve creare una variabile a livello di classe che tenga traccia del valore della proprietà.

```
Private thisVal As String

Public Property Get myProp() As String
   myProp = thisVal
End Property

Public Property Let myProp(ByVal vNewValue As String)
   If vNewValue <> thisVal Then thisVal = vNewValue
End Property
```

Metodi

I metodi di classe vengono implementati semplicemente come funzioni o procedure pubbliche. Per esempio, si può implementare un metodo che collauda la proprietà myProp definita un attimo fa visualizzando una casella di messaggio che include tale proprietà:

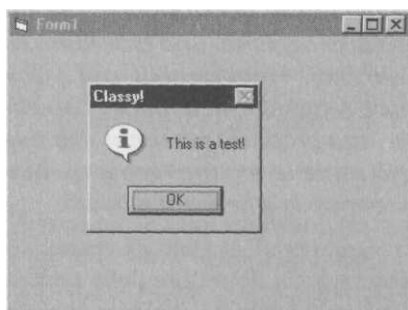
```
Public Function myMeth()
   MsgBox myProp, vbInformation, "Classico!"
End Function
```

Uso delle proprietà e dei metodi di classe

Per invocare la proprietà di classe personalizzata myProp e il metodo personalizzato myMeth, che abbiamo appena visto come si creano, dapprima si deve creare un esemplare di myClass, ricordando l'analogia dello stampino per biscotti e del biscotto. Poi, è possibile assegnare valori di proprietà e chiamare i metodi, usando l'operatore punto. Ecco il codice richiesto, collocato nell'evento Click di un forni:

```
Private Sub Form_Click()
   Dim X As New myClass
   X.myProp = "This is a test!"
   X.myMeth
End Sub
```

Se si esegue il progetto contenente il forni e il modulo myClass, e si fa clic sul form, si otterrà una casella di messaggio come quella mostrata in Figura 3.9.

Figura 3.9 *Si possono facilmente invocare le proprietà e i metodi delle classi, ma prima si deve creare un esemplare detta classe.*

Creazione, scatto e gestione degli eventi personalizzati

Per aggiungere un evento al modulo myClass, si usa l'istruzione Event per dichiarare l'evento con qualsivoglia argomenti. Gli eventi devono essere dichiarati come Public. Si fa scattare l'evento, di nuovo quando si desidera, entro il modulo di classe con l'istruzione RaiseEvent. Bisogna fornire tutti i parametri richiesti,

Come esempio, si potrebbe dire per amore della scienza, aggiungeremo un evento chiamato Frodo al modulo myClass. Frodo viene fatto scattare dall'invocazione di myMeth seguendo la visualizzazione della casella di messaggio myMeth. Il Listato 3.4 mostra il codice revisionato per il modulo myClass con l'ulteriore inserimento dell'evento Frodo.

Listato 3.4 *Aggiunta di un evento.*

```
Option Explicit
Public Event Frodo()
Private thisVal As String

Public Property Get myProp() As String
   myProp = thisVal
End Property

Public Property Let myProp(ByVal vNewValue As String)
   If vNewValue <> thisVal Then thisVal = vNewValue
End Property

Public Function myMeth()
   MsgBox myProp, vbInformation, "Classico!"
   RaiseEvent Frodo
End Function
```

Il passo successivo è accedere al gestore di eventi nel modulo di forni che usa un esemplare di questo modulo di classe.

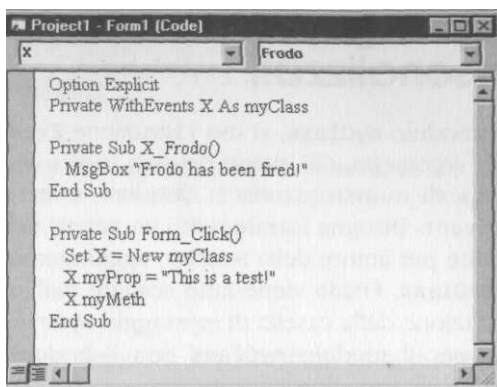
Nella sezione Declarations del modulo di forni, dichiarare una variabile privata del tipo di classe, mediante la parola chiave WithEvents:

```
Private WithEvents X As myClass
```

Adesso si può usare la casella di riepilogo a discesa *Object* per accedere all'oggetto X. Con *X* selezionato nella casella di riepilogo a discesa *Object,* si può accedere al suo evento Frodo nella casella di riepilogo a discesa *Procedure,* come mostrato nella Figura 3.10.

Quando si fa clic sull'evento *Frodo* nella casella di riepilogo a discesa *Procedure,* VB crea il codice modello del gestore di eventi:

```
Private Sub X_Frodo()

End Sub
```

Figura 3.10 *Si può usare il modello di gestore di eventi generato da VB per aggiungere qualsivoglia codice.*

Si può aggiungere qualsivoglia codice a questo evento per collaudare che sia veramente scattato. Per esempio:

```
Private Sub X_Frodo()
   MsgBox "Frodo has been fired!"
End Sub
```

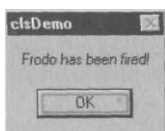
Per creare un esemplare di una variabile oggetto dichiarata usando la parola-chiave WithEvents *non si può usare la sintassi* Dim...As New, *chiamata creazione implicita. Invece, si deve crearla esplicitamente, usando l'istruzione* Set.

Il Listato 3.5 mostra il codice revisionato per il modulo di form che crea un esemplare WithEvents di myClass e risponde allo scatto di Frodo:

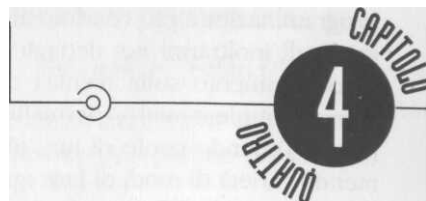
Listato 3.5 *Risposta a un evento personalizzato.*

```
Option Explicit
Private WithEvents X As myClass

Private Sub X_Frodo()
   MsgBox "Frodo has been fired!
End Sub

Private Sub Form_Click()
   Set X = New myClass
   X.myProp = "This is a test!"
   X.myMeth
End Sub
```

Se si esegue il progetto e si fa clic sul form, prima si vedrà la casella di messaggio myMeth. Poi si vedrà la casella di messaggio invocata nel gestore di eventi Frodo, come mostrato nella Figura 3.11.

Figura 3.11 *Una finestra di messaggio invocata da un gestore di eventi.*

Riepilogo

- Abbiamo visto i diversi tipi di file sorgente che entrano in un progetto Visual Basic 6.
- È stata discussa la programmazione guidata da eventi e sono stati introdotti i concetti di evento, proprietà e metodi.
- Abbiamo visto informazioni dettagliate sugli eventi dei form e il loro ordinamento di scatto e come usare la funzione MsgBox.
- Abbiamo visto come aggiungere del codice ai form e ad altri eventi di oggetti.
- È stata fornita una panoramica della programmazione orientata agli oggetti e del posto di VB nel mondo dell'OOP.
- Abbiamo visto una tecnica efficace per creare un passaggio di parametri riutilizzabile, incapsulato, e lascamente accoppiato verso e da un form finestra di dialogo.
- Sono stati spiegati i moduli di classe.
- Abbiamo visto come creare e gestire eventi.
- Abbiamo scoperto come aggiungere moduli e procedure a un progetto e come designare un form o una procedura di avvio.
- Abbiamo mostrato come modificare la proprietà BackColor di un form nel codice o nella finestra *Properties.*
- Abbiamo esplorato una chiamata di metodo che centra un form sullo schermo come alternativa a impostare la proprietà StartUpPosition del form.

SINTASSI DI VISUAL BASIC PER PROGRAMMATORI

- I dettagli di alcune delle più importanti caratteristiche del linguaggio Visual Basic
- La sintassi delle operazioni con gli oggetti
- Chiamare da un programma Visual Basic una procedura che si trova in una libreria esterna
- Chiamare una funzione dell'API di Windows

I programmatori esperti sanno che molto del loro lavoro si riduce a questioni di sintassi di linguaggio. Questo capitolo fornisce le informazioni sulla sintassi che servono a lavorare efficacemente con Visual Basic, purché, naturalmente, ci si sia già imbattuti nei relativi concetti.

Panoramica sulla definizione del linguaggio

Come molti sanno, il linguaggio Visual Basic è un discendente del BASIC (Beginner's All-Purpose Symbolic Instruction Code, cioè codice di istruzioni simboliche ad uso generico per principianti). Le specifiche originarie del BASIC sono state formulate nel 1963 da John G. Kemeny e Thomas Kurtz del Dartmouth College. Kemeny e Kurtz intesero il BASIC come linguaggio di insegnamento. Erano più interessati all'intuitività di utilizzo, nel senso che progettarono il linguaggio in modo da somigliare il più possibile all'inglese, che all'ottimizzare l'elenco di funzionalità e l'implementazione hardware.
Un grave difetto delle prime versioni del linguaggio Basic, dal punto di vista dello sviluppatore serio, era la sua mancanza di *strutturazione.* La progettazione strutturata di programmi è caratterizzata da sistemi che sono divisi in oggetti e da routine che hanno interfacce ridotte e ben definite, i cui dettagli implementativi sono reciprocamente nascosti (vedere le osservazioni sull'accoppiamento nel capitolo precedente). Al contrario, i programmi non strutturati sono resi riconoscibili dall'uso delle istruzioni Goto e dei salti logici a destinazioni specifiche, che producono programmi talvolta indicati dispregiativamente come "codice a spaghetti".

Visual Basic nella sua sesta versione ha fatto una strada molto lunga dalle sue umili radici nel BASIC. Sebbene mantenga molto dell'amichevolezza e facilità d'uso dei Basic precedenti, oggi ha un elenco di funzionalità immensamente ricco e potente con un'incredibile potenzialità di estensione. Adesso è certamente possibile progettare complesse applicazioni Visual Basic in un modo altamente strutturato e rigoroso. Queste caratteristiche hanno aiutato a rendere Visual Basic il linguaggio di programmazione più venduto al mondo.
Prima di inoltrarmi nei dettagli del linguaggio Visual Basic, voglio aggiungere un altro commento sulla tecnica di programmazione VB in generale. Visual Basic, come è implementato, permette allo sviluppatore di fare quasi tutto in qualsiasi modo. Tenendo conto di tutti gli strumenti disponibili di terze parti, si ha una tremenda varietà di modi di fare quasi tutto, tanto che una parte sostanziale del lavoro di sviluppare in VB sta nel mantenersi aggiornati sugli strumenti disponibili. Tutto ciò significa che si possono fare le cose nel modo giusto o nel modo sbagliato. VB non impedisce di progettare malamente i sistemi. Perciò, è fondamentale imparare e prendere l'abitudine di usare tecniche corrette di progettazione e di programmazione. Questa affermazione si applica a molti aspetti della programmazione VB. Per esempio, prendere l'abitudine di usare l'istruzione

```
Option Explicit
```

nei propri progetti a lungo andare fa risparmiare ore di debug. Questa istruzione fa sì che il compilatore renda obbligatoria la dichiarazione esplicita delle variabili e rifiuti la tipizzazione implicita. Si veda la sezione "Usare l'istruzione Option Explicit" un po' oltre nel capitolo.
Seguendo la sua filosofia del "fallo a modo tuo", Visual Basic permette di scrivere del codice che è strutturato, o del codice che è altamente non strutturato. Naturalmente, è difficile, se non impossibile, fare il debug e manutenere il codice non strutturato. Se non si struttura il proprio codice, è molto più difficile essere chiari su ciò che sostanzialmente stia facendo un programma. Forse niente è più importante per diventare un maestro di programmazione Visual Basic che assimilare e usare regolarmente le tecniche che servono a strutturare appropriatamente i propri programmi. Fortunatamente, in VB è facile e naturale scrivere programmi in modo strutturato.

Righe di codice e commenti in Visual Basic

Normalmente, si introduce un'istruzione Visual Basic su una sola riga, il che significa che la fine della riga fisica indica la fine dell'istruzione. Così, si può pensare alla fine della riga come al solito segno implicito di fine-istruzione. Ciò si contrappone all'uso in linguaggi più altamente strutturati. Per esempio, in Object Pascal, generalmente si deve rendere esplicita la fine delle istruzioni con un punto-e-virgola (;).
Essenzialmente non c'è limite alla lunghezza teorica di un'istruzione Visual Basic introdotta in questo modo su una sola riga. Comunque, è una pratica di codifica scadente creare istruzioni più lunghe di una riga del listato o della finestra di codice. Con la finestra di codice dimensionata nel modo in cui normalmente la si

usa si dovrebbe poter leggere ogni istruzione senza dover far scorrere orizzontalmente il codice. La ragione, naturalmente, è la direttiva primaria dello stile di programmazione: "Dovrai scrivere codice chiaro". Un'istruzione, se può essere vista per intero, è molto più chiara ed è meno probabile che contenga degli errori.
C'è un'altra opzione che riguarda la terminazione delle istruzioni VB. Se si desidera mettere su una sola riga più di una istruzione, si può usare un due-punti (:) per separare le istruzioni multiple. Questo esempio combina tre istruzioni su una sola riga:

```
txtFrodo.text = "Not Orc" : MyColor = vbRed : Samwise = "Hobbit"
```

Includere dei commenti all'interno del codice è un modo importante di rendere il proprio lavoro più chiaro agli altri. Commentare il codice può perfino rendere più facile a sé stessi comprenderlo quando vi si ritorna in seguito.
Comunque, il miglior consiglio è creare programmi che siano *auto-documentati.* Questa parola significa, per i conoscitori, che lo scopo del codice dovrebbe essere reso chiaro dal suo progetto strutturale, dalla chiarezza di esposizione delle istruzioni e del controllo di flusso, nonché dalla scelta dei nomi delle variabili. Quando si scrive del codice in questa maniera, si dovrebbero usare commenti solamente per due scopi:

- Per chiarificare qualcosa che non è altrimenti evidente; un esempio potrebbe essere la gamma di valori attesi per un parametro
- Per fornire un'intestazione di routine che indichi l'autore, la data, lo scopo, e l'accoppiamento per un intero progetto, per un modulo, o per una procedura

In questi due contesti, non commentare il codice è una manifestazione di uno dei peccati mortali: la pigrizia. In Visual Basic, si possono creare commenti in due modi:

- Con un singolo apostrofo
- Con la parola riservata Rem

In entrambi i casi, la parte della riga che segue l'indicatore di commento è considerata un commento e viene ignorata dall'interprete o dal compilatore VB.
L'indicatore apostrofo può iniziare una riga, o apparire ovunque all'interno di essa.
La parola riservata Rem deve iniziare una riga o deve essere preceduta da un due-punti, interrompendo la riga come prima descritto.
Tra l'altro, un'anomalia riguardante Rem è che si può saltare a una riga etichettata contenente Rem usando un'istruzione Goto o Gosub. Per ragioni appena accennate sopra in questo capitolo, cioè il "codice a spaghetti", fare in questo modo non è una pratica particolarmente furba.
Ecco degli esempi di ogni stile di commento:

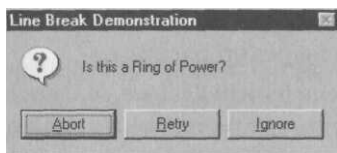
```
'Osservate il contatore sulla prossima riga!
OverFlowCount = NextMule + 1 ' OverFlowCount > hdTop provoca un errore
Rem hdTop è una costante globale attualmente impostata a 20000.
Response = 3: Rem Response = MsgBox (Prompt, Buttons, Title, _
   Helpfile, Helpfile context ID)
```

Spezzare le righe lunghe

In Visual Basic 6, per chiarezza, si possono spezzare le istruzioni lunghe su più righe. La continuazione di un'istruzione sulla riga fisica seguente è indicata da uno spazio seguito dal carattere di sottolineatura (_). Ovviamente, i caratteri di continuazione di riga non devono venire posti all'interno di una stringa letterale. Per esempio:

```
Dim Response As Integer
Response = MsgBox("Is this a Ring of Power?", _
    vbAbortRetryIgnore + vbQuestion, _
    "Line Break Demonstration", _
    "Frodo.Hlp", _
    23)
Rem Response = MsgBox (Prompt, Buttons, Title,
        Helpfile, Helpfile context ID)
```

Questo è molto più facile da leggere di come sarebbe se la funzione MsgBox e il commento che la spiega si distendessero su una sola riga (Figura 4.1).

Figura 4.1 *codice che crea la casella di messaggio di Frodo usa continuazioni di riga.*

L'esempio mostrato nel Listato 4.1 è un buon modo di usare i commenti per creare delle intestazioni di routine. Un trucco usato da molti sviluppatori professionisti è salvare in un piccolo progetto VB i modelli della manciata di intestazioni di routine che probabilmente verranno usati in vari contesti e progetti. Per esempio, un'intestazione di modulo dovrebbe essere in qualche modo diversa da un'intestazione di subroutine o di funzione. Si potrebbero anche salvare le intestazioni in una directory come file di testo con un'estensione .Txt, e usare Blocco note (Notepad) per crearle e modificarle.

Listato 4.1 *Una semplice intestazione di modulo.*

```
'*   Modulo:   DataAccess.Bas
'*   Autore:   Carolina Codifica   Data di creazione:
'*
'*  Questo modulo di codice contiene tutte le routine di accesso
'*  ai dati e utility per tutto il progetto.
'*  Tutte le routine sono per uso interno solamente
'*  tranne:  GetData, PutData, ValidateData.
```

```
'*  Esaminate queste routine per spiegazioni dei parametri.
'*
'*
```

Si può usare il comando Insert File, *che si trova nel menu* Edit, *per aggiungere facilmente al codice il contenuto di un file di testo. Il contenuto del file di testo viene inserito alla posizione del cursore quando la finestra di codice è apena.*

Abbiamo una quantità di modi facili per copiare e incollare i modelli di intestazione nel nostro codice quando ci servono! Uno dei miei principi operativi è di rendere facile seguire le buone abitudini. In tal modo, tendo a metterle in pratica più spesso, traendone vantaggio nel lungo termine.

Gli identificatori, le costanti e le variabili

Gli identificatori vengono impiegati per denominare le cose. Le costanti e le variabili sono cose a cui si deve dare un nome. Le costanti rappresentano valori fissi in un programma, mentre le variabili rappresentano valori variabili.

Gli identificatori

Il termine *identificatore* si riferisce a qualunque elemento di un programma VB a cui è stato dato un nome. Questo include le costanti, le variabili, i nomi di subroutine e i nomi di funzione. Gli identificatori:

- Devono cominciare con una lettera.
- Non possono contenere spazi, punti, o caratteri di dichiarazione di tipo. I caratteri di dichiarazione di tipo devono apparire alla fine del nome.
- Non possono essere parole chiave che sono riservate per il Visual Basic. Per esempio, For non è un identificatore VB valido perché va in conflitto con l'uso riservato dell'istruzione `For...Next`.
- Gli identificatori che si riferiscono a variabili, a costanti e a procedure possono, in teoria, essere lunghi fino a 200 caratteri. Quelli che si riferiscono a controlli, forni, classi, e moduli non possono superare i 40 caratteri. Come buona pratica di codifica, gli identificatori non dovrebbero avere più di circa 25 caratteri. In particolare, si vuole essere in grado di leggere chiaramente i nomi di oggetti e di procedure nello spazio che è disponibile nelle caselle di riepilogo a discesa *Object* e *Procedure* in visualizzazione codice; se si usano nomi troppo lunghi, può diventare difficile.

Le costanti

Le *costanti* sono identificatori usati al posto di valori che si presentano in molti punti del codice, oppure al posto di valori la cui funzionalità non risulterebbe altrimenti chiara. È più facile comprendere a colpo d'occhio il significato di una costante con un nome appropriato che si riferisca, per esempio, al massimo numero

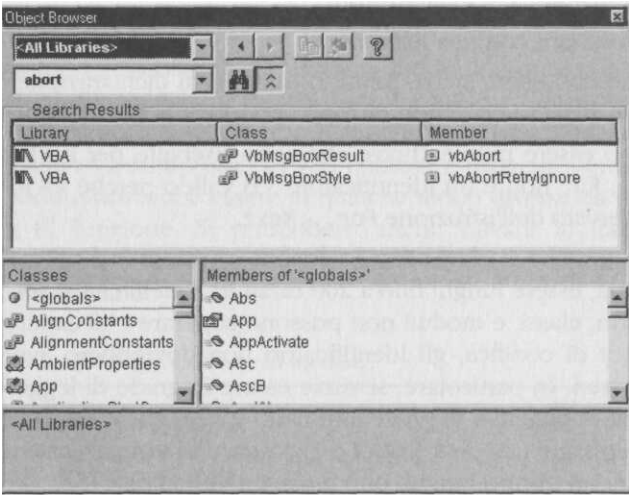
di record in una routine di ricerca, che fare testa o croce del significato di un valore numerico che è l'equivalente della costante. Per esempio:

```
Const MaxSearchRecs = 20964
```

definisce nel codice un valore chiaro. In seguito sarà molto più facile comprendere l'intenzione di un confronto con MaxSearchRecs che non di un confronto con 20964.
Inoltre, usare le costanti rende molto più facile manutenere il codice, se i valori costanti devono venire modificati in seguito. È di particolare aiuto collocare tutte le dichiarazioni di costanti all'inizio di un modulo così che si possa ritoccarle facilmente. La sintassi di molti linguaggi lo richiede espressamente. È bene strutturare i propri progetti VB in questo modo, con blocchi manutenibili di costanti, come valida pratica di codifica.

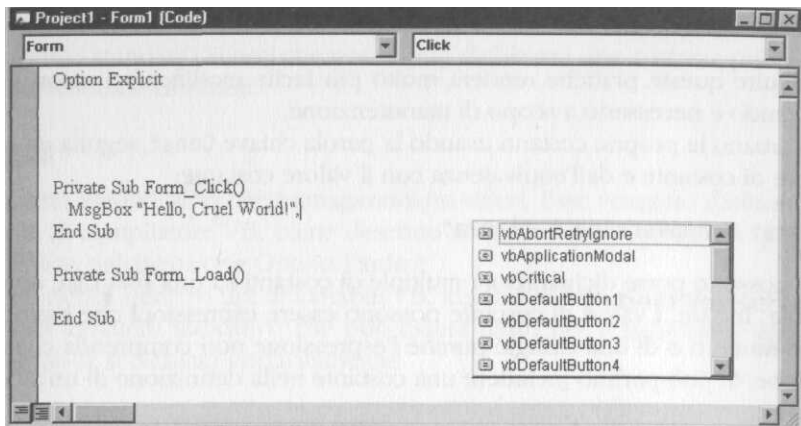
Le costanti predefinite

Visual Basic è fornito di molte costanti predefinite. Queste costanti possono venire usate nel codice senza alcun tipo di dichiarazione. Un buon modo di trovare queste costanti è di usare l'Object Browser. Se si introduce del testo nel campo di ricerca dell'Object Browser, vengono trovate le relative costanti. Per esempio, se si introduce la parola "abort", il *Browser presenta,* due costanti rilevanti, come mostrato in Figura 4.2: la costante vbAbort che è un membro della classe vbMsgBoxResult della libreria VBA, e la costante vbAbortRetrylgnore che è un membro della classe vbMsgBoxStyle della libreria VBA.

Figura 4.2
Si può usare l'Object Browser per trovare rapidamente le costanti predefinite disponibili.

Tra l'altro, quando si seleziona una costante nell'Object Browser, viene indicato l'equivalente numerico della costante.

Si può usare la finestra di codice di VB6 per accertare facilmente le costanti predefinite disponibili, e poi aggiungerle al proprio codice. Per esempio, dopo aver introdotto il primo argomento di un'istruzione MsgBox, si può selezionare *List Constants*

dal menu di scelta rapida della finestra di codice per visualizzare una casella di riepilogo a discesa delle costanti che sono sintatticamente corrette per il contesto, come mostrato nella Figura 4.3. Facendo doppio clic su una delle costanti, la si aggiunge all'istruzione MsgBox.

Figura 4.3
Sipuò usare il comando List Constante della finestra di codice per trovare facilmente le costanti predefinite appropriate.

Sebbene in realtà si possano sempre usare i valori invece delle costanti predefinite, è una buona idea programmare con le costanti, perché rendono molto più chiaro ciò che i valori rappresentano.
Le costanti delle librerie di Visual Basic e di Visual Basic for Applications iniziano con le lettere "vb." Per esempio:

```
vbShapeRoundedRectangle '=4, dalla libreria VB ShapeConstants,
  'che indica un rettangolo arrotondato con curve

vbFormFeed  'Carattere ASCII 12, equivalente a Chr$(12), dalla
 'libreria di costanti VBA, usato per forzare un avanzamento carta.
```

Nelle versioni di VB precedenti a Visual Basic 4, le costanti predefinite erano denominate con tutte le lettere maiuscole e le sottolineature, come in TILE_VERTICAL. L'equivalente in VB6 è vbTileVertical. Le costanti nel vecchio stile sono state mantenute per compatibilita all'indietro.

Le costanti definite dall'utente

Come già trattato in questo capitolo, è buona pratica di programmazione usare delle costanti al posto dei valori che verrebbero usati ripetutamente. La ragione principale è che, nel contesto di un'applicazione, è facile determinare il significato di una costante ben denominata.

Le costanti dovrebbero venire definite tutte in un solo posto. Le costanti globali dovrebbero venire dichiarate in un solo modulo, non sparpagliate in numerosi moduli. Le costanti a livello di modulo dovrebbero andare all'inizio della sezione Declarations *del modulo. Le costanti a livello di procedura dovrebbero venire poste all'inizio del codice della procedura. Tutte le costanti il cui significato non è perfettamente chiaro dai loro nomi dovrebbero essere commentate.*

Se le costanti sono sparpagliate in diversi moduli e procedure, si possono iniziare tutti i nomi di costante con un identificatore univoco, per esempio le proprie iniziali, per renderepiùfacile trovarle. Per esempio:

```
Const hdTop
Const hdMaxPrice
```

Seguire queste pratiche renderà molto più facile modificare i valori delle costanti quando è necessario a scopo di manutenzione.
Si creano le proprie costanti usando la parola chiave Const seguita da un identificatore di costante e dall'equivalenza con il valore costante:

```
Const MyAppVersion = "1.02a"
```

Si possono Forre dichiarazioni multiple di costanti su una sola riga, separate da virgole. Inoltre, i valori di costante possono essere espressioni che hanno il valore di un numero o di una stringa, purché l'espressione non comprenda chiamate di funzione. Si può perfino includere una costante nella definizione di un'altra. Per esempio:

```
Const MyAppName = "Bear Games"
Const NameAndVersion = MyAppName & ", Version: " & MyAppVersion
```

Quando si definiscono costanti che usano altre costanti, si deve stare attenti a evitare riferimenti circolari tra le costanti, in cui due o più costanti sono definite ognuna in termini delle altre.
Si può tipizzare esplicitamente le costanti usando la direttiva As:

```
Const NewApp As String = "Panther Games"
```

Per ragioni che tratteremo nella prossima sezione, la tipizzazione esplicita è solitamente una buona idea.
Un'osservazione finale è che le costanti definite dall'utente hanno un *ambito d'azione* (scope), proprio come le variabili. Si può pensare all'ambito d'azione come all'estensione fino a cui le variabili o le costanti sono visibili e possono essere referenziate all'interno di un programma. Una variabile o costante che è visibile solamente in una procedura ha un ambito d'azione limitato, mentre una che può venire usata in tutto un programma è di ambito d'azione ampio. L'ambito d'azione viene trattato in maggiore dettaglio più avanti in questo capitolo e nel Capitolo 13.
Opzionalmente, si può aggiungere una parola chiave Private o Public all'inizio di una dichiarazione di costante. Per esempio:

```
Public Const CryptKey = "A56789C"
Private Const UltimateAnswer = 42
```

Usando queste parole chiave, le regole dell'ambito d'azione delle costanti sono le seguenti:

- Per dichiarare una costante locale a una procedura, si deve dichiararla nella procedura senza parole chiave o con la parola chiave Private.

- Per dichiarare una costante locale a un modulo, nel senso che è disponibile a tutte le procedure del modulo, ma non a quelle di altri moduli, si deve dichiararla nella sezione General Declarations del modulo senza parole chiave o con la parola chiave Private.
- Per rendere una costante disponibile globalmente, si deve dichiararla con la parola chiave Public nella sezione General Declarations di un form o di un modulo standard. Si noti che non si può dichiarare una costante pubblica in un modulo di classe.

Le variabili

Le *variabili* sono identificatori che immagazzinano valori. Esse vengono *dichiarate,* cioè rese note al compilatore VB, come descritto nella prossima sezione di questo capitolo, "Utilizzo dell'istruzione Option Explicit".
La Tabella 4.1 elenca i diversi tipi di variabili VB, insieme al loro contenuto consentito, e il tipo di carattere indicativo che può essere usato per il particolare tipo di variabile quando la si dichiara implicitamente.

Tabella 4.1 *I diversi tipi di variabili Visual Basic.*

Tipo	Dimensione in memoria	Contenuto	Carattere identificativo
Byte	1 byte	Numerico: da O a 255	Nessuno
Boolean	2 byte	Logico: True o False	Nessuno
Integer (intero)	2 byte	Numerico: da -32.768 a 32.767	%
Long (intero lungo)	4 byte	Numerico: da -2.147.483.648 a 2.147.483.647	&
Single (a virgola mobile a precisione singola)	4 byte	Numerico: da -3,402823E38 a -l,401298E-45 per valori negativi; da l,401298E-45 a 3,402823E38 per valori positivi	!
Doublé (a virgola mobile a precisione doppia)	8 byte	Numerico: da -1,79769313486232E308 a -4,94065645841247E-324 per valori negativi; da 4,94065645841247E-324 a 1,79769313486232E308 per valori positivi	#
Currency (intero scalato)	8 byte	da-922.337.203.685.477,5808 a 922.337.203-685.477,5807	@
Date (data)	8 byte	Dal 1 gennaio 100 al 31 dicembre 9999	Nessuno
Object	4 byte	Qualunque riferimento ad oggetto (in realtà è un puntatore ad esemplare di un oggetto)	Nessuno

(continua)

Tabella 4.1 *I diversi tipi di variabili Visual Basic,* *(continua)*

String (a lunghezza variabile)	10 byte più la lunghezza della stringa	da O ad approssimativamente 2 miliardi di caratteri (contro i circa 65.400 delle versioni per Microsoft Windows 3)	$
String (a lunghezza fissa)	Lunghezza della stringa	da 1 ad approssimativamente 65.400 caratteri; Dim Frodo As String * 20 è un esempio di dichiarazione di una stringa di lunghezza fissa contenente 20 caratteri	$
Variant (con numeri)	16 byte	Qualunque valore numerico fino alla gamma di un Doublé	Nessuno
Variant (con caratteri)	22 byte più la lunghezza della stringa	La stessa gamma di una stringa di lunghezza variabile	Nessuno
Definito dall'utente (usando Type)	La somma delle dimensioni richieste dagli elementi	La gamma di ogni elemento è quella del suo tipo di dati	Nessuno

Come molti sapranno, in un progetto possono coesistere delle variabili con lo stesso nome ma diversi ambiti d'azione. Per esempio, non c'è certamente nessuna ragione perché non si possano avere due variabili locali dichiarate

```
Private Pippin As Long
```

in una procedura di un modulo e

```
Private Pippin As String
```

in un'altra. Le due variabili Pippin sono, naturalmente, diverse e completamente indipendenti.

La questione diventa un po' più complessa quando si hanno più variabili con lo stesso nome e ambito d'azione sovrapposto. Per ulteriori informazioni su questo argomento, vedere il Capitolo 13 di questo libro.

Oltre all'ambito d'azione, le variabili hanno una *durata di vita.* La durata di vita di una variabile è il periodo per cui mantiene i suoi valori. Di default, le variabili a livello di modulo e a livello pubblico in VB *persistono,* cioè vivono, per tutto il tempo per cui l'applicazione è viva e caricata in memoria. Le variabili Private (locali), d'altra parte, persistono solamente fin tanto che la procedura in cui si trovano è in esecuzione.

Le parole chiave Public e Private

Le variabili possono avere un ambito d'azione tale da essere locali alle procedure, disponibili a tutte le procedure di un modulo, o disponibili globalmente a tutti i moduli. Ecco come funziona.

- Le variabili dichiarate in una procedura sono locali a quella procedura. All'interno di una procedura, le seguenti due dichiarazioni sono equivalenti:

```
Private Meriadoc As String
Dim Meriadoc As String
```

- Non si può usare la parola chiave Public nelle dichiarazioni interne a una procedura.
- Le variabili dichiarate nella sezione General Declarations di un modulo con la parola chiave Private sono disponibili a tutte le procedure del modulo ma non ad altri moduli. Dichiarare una variabile a livello di modulo usando Private è esattamente equivalente nell'impatto sull'ambito d'azione a dichiararla con Dim.
- Le variabili dichiarate nella sezione General Declarations di un modulo usando la parola chiave Public sono disponibili a tutte le procedure di tutti i moduli. Fanno eccezione i moduli di classe, nei quali non è consentito dichiarare variabili Public. Per esempio:

```
Public MyObject As New Form1
```

- Si noti che la parola chiave Global, usata fino a VB3 allo stesso fine di Public, è stata mantenuta per scopi di compatibilita all'indietro.

Per fare in modo che le variabili private persistano anche quando la procedura in cui si trovano non è più in esecuzione, cioè per *preservare* il loro valore, si usi la parola chiave Static (statico) al posto di Dim o di Private. Ecco una funzione che usa una dichiarazione statica per conservare un contatore:

```
Function TestStatic(NumVal As Integer)
    Static TestCount As Integer
    TestCount = TestCount + NumVal
    TestStatic = TestCount
End Function
```

Si può collaudare questa funzione chiamandola ripetutamente, per esempio, da un gestore di clic del forai:

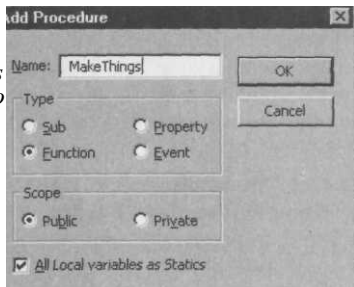
```
Private Sub Form_Click()

    MsgBox Str(TestStatic(1))
End Sub
```

Se si esegue questo codice, si vedrà che il valore restituito si incrementa di 1 ogni volta che si chiama TestStatic. Al contrario, se TestCount fosse stata dichiarata come non persistente usando Dim o Private, la funzione TestStatic restituirebbe 1 ogni volta, senza mai incrementarsi.
Per rendere statiche tutte le variabili locali di una procedura, si aggiunge la parola chiave Static all'intestazione della procedura. Per esempio:

```
Static Function TestStatic(NumVal As Integer)
```

Si possono creare automaticamente procedure e funzioni le cui variabili locali sono tutte statiche impostando l'opzione MI Locai Variables as Statics *nella finestra di dialogo* Add Procedure *(vedere la Figura 4.4).*

Figura 4.4

Se si seleziona All Local Variables as Statics quando si usa la finestra di dialogo Add Procedure, la procedura viene definita con la parola chiave Static.

Le variabili varianti

Se si dichiara implicitamente una variabile senza Forre alla fine del suo nome un carattere di dichiarazione di tipo, VB assume che sia una variante. Ecco due esempi di uso implicito non variante:

```
FlukeCount% = 1  'FlukeCount è un intero
HobbitName$ = "Frodo"  'HobbitName è una stringa
```

Ecco un uso implicito di variante (si noti che la stessa variabile immagazzina sia stringhe che numeri):

```
WhatEver = "I like to sing!"
WhatEver = 42
```

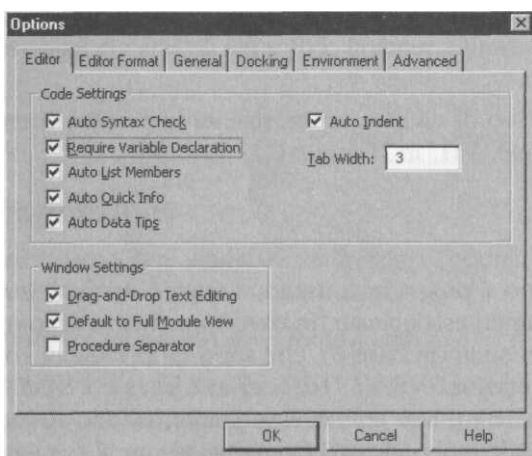
Non si deve pensare a una variabile variante come a una variabile senza tipo. Piuttosto, una variante è una variabile capace di assumere vari tipi e che in generale converte automaticamente i suoi valori immagazzinati fra questi tipi. VB incapsula l'immagazzinamento interno delle varianti e può automaticamente modificare il tipo della variante che è stato immagazzinato. È imFortante sapere che gli oggetti, tra cui i form, i controlli, e gli oggetti di automazione OLE, possono venire immagazzinati in una variabile variante. In effetti, quello che viene immagazzinato è solo un puntatore all'oggetto!

Utilizzo dell'istruzione Option Explicit

Le variabili possono essere dichiarate *implicitamente o esplicitamente.* Si dichiara implicitamente una variabile semplicemente usando la variabile nel codice.
Normalmente si dovrebbe rendere obbligatoria la dichiarazione esplicita delle variabili come buona pratica di codifica per aiutare a minimizzare gli errori dovuti ai refusi nei nomi delle variabili. La dichiarazione esplicita delle variabili diventa obbligatoria in un modulo quando si aggiunge l'istruzione

```
Option Explicit
```

nella sezione Declarations di una classe, di un form, o di un modulo standard. Si può impostare l'IDE in modo che aggiunga automaticamente la dichiarazione Option Explicit in ogni nuovo modulo, e suggerisco di farlo (vedere la Figura 4.5), ma la dovrete aggiungere manualmente ad ogni modulo preesistente. Scegliete *Options* dal menu *Tools* e assicuratevi che nella scheda *Editar* della finestra di dialogo *Options* sia impostata l'opzione *Require Variable Declaration.*

Figura 4.5 *Impostare la casella di controtto Require Variable Declaration nella scheda Editar della finestra di dialogo Options per rendere obbligatoria la dichiarazione esplicita di variabili in tutti i nuovi moduli.*

Dopo aver reso obbligatoria la dichiarazione esplicita, si possono usare solamente le variabili che sono state precedentemente dichiarate usando le istruzioni Dim, Private, Public, o Static. Se si tenta di usare una variabile che non è ancora stata dichiarata con Dim o con un'altra parola chiave, si otterrà un messaggio di errore dal compilatore quando si tenta di mandare in esecuzione il codice sorgente. Ecco alcuni esempi di dichiarazioni esplicite:

```
Option Explicit
Dim Whatever   'Variante
Dim OrcName As String 'Stringa
Dim BodyType As Integer 'Intero
Dim MyName As String, HobbitName As String '2 stringhe
Dim EntName As String, Dim IdNum As Long 'stringa e long
Dim X As New Form1 'puntatore a un'istanza di Form1
```

Usare le variabili dichiarate implicitamente può essere comodo. Il problema è che è troppo facile introdurre una nuova variabile digitando un nome leggermente diverso dal nome di una variabile esistente. L'intenzione era di modificare il contenuto di una variabile, invece se ne crea una nuova. Per esempio, TempVal, TempV1, TemVal, e Tempvall appaiono tutte sorprendentemente simili ad un'occhiata veloce. L'individuazione dei possibili bachi risultanti dalla confusione fra questi nomi di variabile leggermente diversi, che non si sarebbero mai presentati se fosse stata obbligatoria la dichiarazione esplicita, può essere difficile e può richiedere molto tempo. Lo sviluppatore serio deve sfruttare tutte le possibilità favorevoli e dichiarare esplicitamente!

Tra l'altro, supponiamo che TempVal, TempV1, TemVal, e Tempvall *fossero* state intese come variabili distinte e fossero state dichiarate esplicitamente. Sebbene non ci sia niente di tecnicamente sbagliato in ciò, si tratta di una cattiva pratica di denominazione delle variabili per tre ragioni:

- Questi nomi non forniscono reali informazioni sul contenuto delle variabili.
- I nomi sono troppo simili tra di loro. Anche se questo non conduce ad un baco come appena descritto, rende il codice più difficile da leggere e meno chiaro.
- Si deve evitare il rischio di digitare erroneamente le parole contenute nei nomi di variabile, come "Val" in "TempVal".

I numeri

La maggior parte del tempo si programma usando i numeri decimali (in base 10). Si possono anche usare i numeri esadecimali (in base 16), che sono rappresentati con il prefisso &H, e i numeri ottali (in base 8), che sono rappresentati con il prefisso &O. Per esempio, nella notazione VB, 255 (decimale) è uguale a &O377 (ottale) ed è uguale a &HFF (esadecimale). Una tecnica imFortante quando si vuole usare un numero ottale o esadecimale molto elevato è di concatenare al numero il carattere di tipo Long (&). Questa strategia fa sì che il numero sia immagazzinato correttamente come intero lungo. Per esempio, per immagazzinare &HFF10F9CC come intero lungo, lo si deve introdurre così:

```
&HFF10F9CC&
```

Gli operatori

Visual Basic ha quattro categorie generali di operatori: aritmetici, di concatenazione di stringhe, di confronto, e logici. La Tabella 4.2 elenca queste categorie con i loro elementi.

Tabella 4.2 *La precedenza e le categorie degli operatori di Visual Basic.*

Aritmetici	Di concatenazione fra stringhe	Di confronto	Logici
Elevamento a potenza (^{A})	Concatenazione di stringa (&) o (+)	Uguaglianza (=) (da non confondere con l'assegnamento)	Not
Negazione (-),		Disuguaglianza (<>)	And
Moltiplicazione e divisione (*,/)		Minore di (<)	Or
Divisione intera (\)		Maggiore di (>)	Xor (esclusione logica)
Resto di divisione (Mod)	Minore o uguale a (<=)	Eqv (equivalenza logica)	
Addizione e sottrazione (+,-)		Maggiore o uguale a (>=)	Imp (implicazione logica)
		Like (usato per confrontare due stringhe usando la corrispondenza tra schemi)	
		Is (usato per confrontare l'equivalenza di due riferimenti ad oggetti)	

L'operatore punto

Sicuramente, è bene fare amicizia anche con l'operatore punto (.). L'operatore punto viene usato:

- Per connettere gli oggetti con i loro oggetti figli, come i controlli
- Per connettere gli oggetti con le loro proprietà e i loro metodi
- Per recuperare e immagazzinare i valori delle variabili che sono state definite entro una struttura definita dall'utente

Per esempio, si può usare l'operatore punto per connettere le proprietà, i controlli, e i form:

```
Form1.BackColor = vbRed 'imposta la proprietà BackColor a rosso
```

e

```
Form1.txtUserld.text = "Finnegans Wake"
'imposta la proprietà testo del controllo txtUserld, che si trova
'sull'oggetto Form1, alla stringa data
```

Se in un assegnamento che usa l'operatore punto, si omette la proprietà del controllo indicato, VB, se può, userà la proprietà di default di quell'oggetto. Per esempio, .Caption è la proprietà di default di un controllo Label. Il seguente codice assegna il valore di Labell.Caption a una variabile; se non c'è nessuna didascalia, viene assegnata una stringa vuota:

```
Dim Contents As String
Contenta  = Form1.Labell
```

In modo simile, Form1.Labell, naturalmente, si riferisce a quella Labell che è figlia (pensiamola come ancora viva) di Form1, mentre Form2.Labell si riferisce a quella Labell che è figlia di Form2. Per esempio, il seguente codice,

```
Form1.ZOrder    1
```

usa l'operatore punto per invocare il metodo ZOrder del form. E

```
Form1.Textl.Move0,0
```

sposta Text1 all'angolo superiore sinistro dell'area cliente di Form1.
Infine, se si ha una struttura definita dall'utente come

```
Type Animai
   Name As String
   Weight As Integer
End Type
```

si può usare l'operatore punto per assegnare e recuperare i valori dalla struttura:

```
Animal.Name =  "Bulgy  Bear"
Size% =  Animal.Weight  'Dimensioni dichiarate implicitamente. Male!
```

Tratteremo più dettagliatamente le strutture definite dall'utente più avanti in questo capitolo, sotto "Le strutture definite dal programmatore".

L'operatore di assegnamento

Visual Basic usa il segno di uguale (=) sia come operatore di confronto che come operatore di assegnamento. Quando è usato come operatore di confronto, = verifica l'uguaglianza; quando è usato per l'assegnamento, come negli esempi precedenti, = trasferisce un valore dal lato destro del segno di uguale all'identificatore sul lato sinistro.
Quando si guarda un'istruzione VB, è imFortante comprendere quale uso dell'operatore = è inteso. Per evitare confusione, alcuni altri linguaggi usano operatori diversi per il confronto e per l'assegnamento. Così, in Object Pascal := significa assegnamento e = significa confronto, mentre in C = significa assegnamento e == (due segni di uguale) significa confronto.
La separazione delle due funzioni ha dei vantaggi, tuttavia, in favore di VB si potrebbe sostenere che è più rapido introdurre semplicemente un segno di uguale invece di due caratteri, e che il contesto solitamente rende chiaro che cosa stia succedendo.

L'operatore di insieme

Tutti i membri di un *oggetto insieme (collection object)* sono a loro volta degli oggetti; per esempio, i controlli di un form. L'operatore di insieme "!" viene usato per fare riferimento a specifici membri di un insieme. Per esempio,

```
Form1.Controls!Label1
```

indirizza il membro Label1 dell'insieme Form1 .Controls. Ci sono due altri modi equivalenti di indirizzare una collezione di controlli senza usare l'operatore "!": si può usare direttamente il nome dell'oggetto, o si può usare il numero indice del membro dell'insieme. Se il controllo chiamato Label1 è il primo membro dell'insieme Form1 .Controls (con un valore di indice di 0), si può accedervi come segue:

```
Form1.Controls("Label1")
Form!.Controls(0)
```

Si troverà altro materiale sulla sintassi del riferimento agli oggetti e agli insiemi di oggetti più avanti in questo capitolo nella sezione "Parlare il linguaggio degli oggetti", e nella Parte III di questo libro.
Come si sarà notato nella Tabella 4.1, "!" è usato anche per indicare che una variabile è di tipo Single. Ci si assicuri di essere chiari riguardo a ciò che "!" stia facendo in un dato contesto, e di evitare confusioni.

La precedenza degli operatori

La Tabella 4.2 ha presentato la maggior parte degli operatori comuni in modo che andando da sinistra a destra e dall'alto in basso si segua l'ordine di precedenza, nel senso che quelli nella colonna più a sinistra vengono valutati per primi, e così via. All'interno di ogni categoria, la precedenza va dall'alto in basso. Si può modificare l'ordine di valutazione aggiungendo delle parentesi a un'espressione. Le operazioni tra parentesi vengono eseguite sempre prima di quelle fuori dalle parentesi. La maggior parte delle funzionalità degli operatori in questa tabella dovrebbero essere abbastanza chiare; se non lo sono, sono tutti spiegati molto bene nella Guida in linea di VB.

La concatenazione fra stringhe

Concatenare due stringhe significa creare una terza stringa che consiste della prima stringa più la seconda stringa. Le stringhe si concatenano usando l'operatore di concatenazione fra stringhe, rappresentato o da una e-commerciale ("&") o da un segno più ("+"). Ecco un esempio:

```
Private Sub Form_Click()
    Dim Msg As String
    Msg = "Tomorrow"
    Msg = Msg & vbCrLf & "   is"
    Msg = Msg & vbCrLf & "another"
    Msg = Msg & vbCrLf & "  DAY!"
    MsgBox Msg
End Sub
```

L'esempio crea il valore di stringa della variabile Msg concatenando ripetutamente nuove stringhe a sé stessa. Un trucco usato qui è l'uso della costante predefinita vbCrLf, il cui valore è la concatenazione della coppia di caratteri aventi codice

ASCIIrispettivamente 13 e 10. vbCrLf è concatenato nella stringa Msg per creare le interruzioni di riga. Infine, la funzione MsgBox viene chiamata per visualizzare la stringa (vedere la Figura 4.6).

Figura 4.6
Ecco la stringa concatenata con le interruzioni di riga generate dal codice precedente.

È imFortante saper concatenare le stringhe per poterle manipolare fluidamente. Una delle grandi forze di Visual Basic è la sua incredibile facilità e ricchezza di funzioni per manipolare le stringhe. Mostreremo altre cose sulla manipolazione di stringhe man mano che procederemo in questo libro. Sebbene molti dei progetti dimostrativi in questo libro contengano tecniche utili per le stringhe, i metodi specifici per la manipolazione di stringhe sono trattati nel Capitolo 13-

I cicli di controllo e le istruzioni condizionali

I cicli di controllo e le istruzioni condizionali abilitano a manipolare l'ordine in cui le istruzioni del programma vengono eseguite. Senza queste istruzioni, il *flusso* del programma, cioè l'ordine in cui le istruzioni del programma vengono eseguite, sarebbe immutabile. Il flusso sarebbe dall'alto in basso e da sinistra a destra, il che sarebbe inadeguato eccetto che per i programmi più semplici.

Le istruzioni If

Le istruzioni If vengono usate per eseguire delle istruzioni condizionatamente, a seconda della valutazione di un'espressione. L'espressione di test deve valere True (vero) o False (falso), ed è spesso un confronto. Le espressioni numeriche soddisfano questa condizione perché VB considera False il valore numerico zero e True ogni valore diverso da zero.
Sono possibili tre tipi di istruzioni If:

- L'istruzione If a "singola riga", in cui l'istruzione viene eseguita se la condizione vale True.
- L'istruzione If "a più righe", in cui un blocco di istruzioni viene eseguito se la condizione è True. Il blocco delle istruzioni viene concluso dalle parole chiave End If.
- L'istruzione If "a più righe", eventualmente contenente molti blocchi di istruzioni. Il controllo di flusso è eseguito tramite la valutazione dell'espressione originale e tramite la valutazione delle clausole opzionali ElseIf e Else.

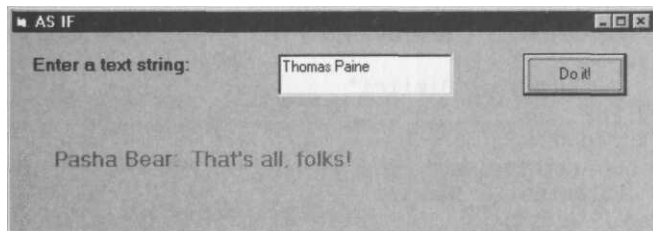
Il Listato 4.2 mostra un esempio che usa i tre tipi di istruzione If. Dimostra anche l'uso di tre funzioni di manipolazione di stringhe: Left, Len e UCase. Left restituisce dei

caratteri dalla parte sinistra di una stringa, Len restituisce la lunghezza di una stringa e UCase restituisce la stringa passata, con le lettere rese maiuscole. Per trovare altre informazioni su queste e altre funzioni di stringa, le si può cercare nella Guida in linea di VB.

Listato 4.2 *Tre tipi di istruzioni If.*

```
private Sub cmdDoIt_Click()
    lblOutput = ""
    If Left(UCase(txtUserInput), 1) = "A" Then
        If Len(txtUserInput) < 4 Then
            lblOutput = "Smerdyakov"
        Else
            lblOutput = "Dimitri"
        End If
    ElseIf (Left(UCase(txtUserInput), 1) > "A") And _
        (Left(UCase(txtUserInput), 1) < "D") Then
        If Len(txtUserInput) < 3 Then
            lblOutput = "Alexei"
        Else
            lblOutput = "Raskolnikov"
        End If
    ElseIf (Left(UCase(txtUserInput), 1) > "E") And _
        (Left(UCase(txtUserInput), 1) < "Q") Then
        If Len(txtUserInput) < 2 Then
            lblOutput = "Fyodor"
        End If
    Else
        If Left(UCase(txtUserInput), 1) = "T"
            Then lblOutput = "Pasha Bear: "
        lblOutput = lblOutput & " That's all, folks!"
    End If
End Sub
```

Per iniziare questo progetto, aggiungere a un form una casella di testo chiamata txtUserInput, un'etichetta chiamata lblOutput, e un pulsante di comando chiamato cmdDoIt. Si noti che il codice usa txtUserInput per far riferimento a txtUserInput.Text, che è la proprietà di default di questo controllo; analogamente, viene usato lblOutput per far riferimento a lblOutput.Caption.
Il programma emette un messaggio nella didascalia di lblOutput quando viene fatto clic sul pulsante di comando, a seconda della prima lettera e della lunghezza dell'input in txtUserInput (vedere la Figura 4.7). Il codice mostrato nel Listato 4.2 va aggiunto al gestore dell'evento Click di cmdDoIt.

Figura 4.7 *Ecco l'output generato dalle istruzioni If quando l'utente inserisce una stringa che inizia con "T".*

le istruzioni If di cmdDoIt possono apparire complesse a un lettore occasionale, sebbene la funzionalità che incapsulano sia davvero molto immediata e non particolarmente sofisticata. Le strutture di controllo annidate possono diventare molto più complicate, e VB non limita i livelli di annidamento ammessi. Un modo imFortante di semplificare le strutture risultanti è di collocare singole chiamate di procedura nelle aree di esecuzione invece di includere istruzioni di esecuzione multiple entro una diramazione di una struttura di controllo. La procedura che è chiamata dal flusso di esecuzione può quindi contenere tutte le istruzioni necessarie per la logica di programma.
Un'altra tecnica che può semplificare le strutture di decisione è l'utilizzo della struttura Select...Case....Else come alternativa alle strutture If. La struttura Select...Case funziona con un'unica espressione di prova che viene valutata una sola volta all'inizio della struttura. Il Listato 4.3 mostra l'esempio precedente sulle strutture If riscritto usando la struttura Select. Penso che sia molto più facile vedere la logica di controllo del flusso quando la struttura è scritta in questo modo! Naturalmente, niente impedisce di sostituire le istruzioni If interne con istruzioni Case annidate. Ho aggiunto questo codice all'evento Click del form in modo che possa venire eseguito nello stesso progetto d'esempio del codice precedente.

Prima creare un'intelaiatura di strutture di controllo

Se prima si creano le corrette istruzioni di controllo, è meno probabile che le strutture di controllo prodotte siano sintatticamente errate, o peggio che siano corrette in termini di sintassi, ma non facciano quello che ci si aspetterebbe. Successivamente, si possono aggiungere semplici istruzioni, per esempio usando la funzione MsgBox, per assicurarsi che il flusso proceda correttamente in base alle espressioni di test. Solamente dopo che si è soddisfatti dell'intelaiatura, si dovrebbero aggiungere le effettive istruzioni di esecuzione.

Listato 4.3 *Revisione dell'evento Click con strutture Select.*

```
Private Sub Form_Click()
    IblOutput = ""
    Dim TestLet As String
    TestLet = Left(UCase(txtUserInput), 1)
    Select Case TestLet
        Case "A"
            If Len(txtUserlnput) < 4 Then
                IblOutput = "Smerdyakov"
            Else
                IblOutput = "Dimitri"
            End If
        Case "B" To "D"
            If Len(txtUserlnput) < 3 Then
                IblOutput = "Alexei"
            Else
```

```
            IblOutput = "Raskolnikov"
        End If
    Case "E" To "Q"
        If Len(txtUserlnput) < 2 Then
            IblOutput = "Fyodor"
        End If
    Case Else
        If TestLet = "T"
            Then IblOutput = "Pasha Bear: "
        IblOutput = IblOutput & " That's all, folks!"
    EndSelect
End Sub
```

Un altro trucco di Visual Basic, che aiuta a chiarificare come il flusso di selezione dipenda dall'input dell'utente, consiste nel chiamare una subroutine passandole un parametro che indichi la selezione dell'utente. Un'istruzione Select. . .Case nella routine chiamata può allora venire usata per eseguire le istruzioni appropriate. (Spesso, la stessa struttura di decisione nella subroutine chiamata dovrebbe avere l'unico scopo di chiamare delle routine dal nome appropriato.)
Per dimostrare questa tecnica, si aggiunga a un forni una casella di riepilogo di nome lstSelect e un pulsante di comando di nome cmdPass. Nell'evento di caricamento del form di avvio (Form_Load), si usi il metodo AddItem della casella di riepilogo per popolare la casella di riepilogo con una serie di possibili scelte per l'utente. Si noti che, affinchè l'esempio funzioni, la prima lettera di ogni selezione deve essere univoca. Ci sono, naturalmente, molti modi di modificare il passaggio del parametro nel caso questa condizione fosse troppo restrittiva. Si potrebbero usare le prime due lettere, anziché solo la prima; oppure si potrebbe usare l'intera stringa o il numero che indica la posizione nella casella di riepilogo.

Siccome la proprietà Sorted *di* lstSelect *è impostata a* True, *l'elenco viene presentato in ordine alfabetico indipendentemente dall'ordine in cui si aggiungono le voci.*

```
Private Sub Form_Load()
    lstSelect.AddItem"Biscotti"
    lstSelect.AddItem "Toast"
    lstSelect.AddItem "Maiale in casseruola"
    lstSelect.AddItem "Grana"
    lstSelect.AddItem "Frutti di bosco"
End Sub
```

Successivamente, si aggiunga all'evento Click del pulsante di comando il codice mostrato nel Listato 4.4. Tale codice verifica se è stata selezionata una voce, poi chiama la subroutine DoStuff contenente la struttura di decisione, passandole l'appropriato parametro.

Listato 4.4 *Verifica della selezione di una voce.*

```
Private Sub cmdPass_Click()
    Dim Choice As String
    If lstSelect.ListIndex = -1 Then
```

```
        'Verifica se è stato selezionato qualcosa
        MsgBox "Nothing is Selected!"
    Else
        Choice = lstSelect.List(lstSelect.ListIndex)
                'Prende la stringa selezionata
        DoStuff Left(UCase(Choice), 1)
        'Chiama la struttura di decisione con la prima
        'lettera della stringa come parametro.
    'In questo modo fa l'assunzione che le varie stringhe
        'abbiano tutte la prima lettera diversa.
    'La chiamata a UCase non è in realtà necessaria perché
        'ogni stringa è stata aggiunta all'elenco comunque
        'con la prima lettera maiuscola, ma perché non avere
        'un po' di ridondanza nelle protezioni?
    End If
End Sub
```

Restituire in una casella di riepilogo il valore di stringa di una voce selezionata

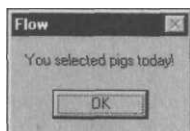
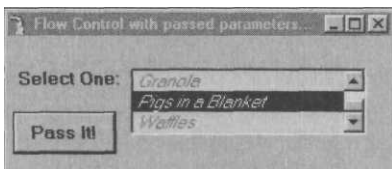
Nella procedura cmdPass mostrata nel Listato 4.4, l'istruzione

```
Choice = lstSelect.List(lstSelect.ListIndex)
```

restituisce il valore di stringa della voce selezionata in lstSelect. Questo è il modo generale di recuperare il contenuto della selezione corrente in una casella di riepilogo normale (ListBox) o combinata (ComboBox), in cui ListorComboObject va sostituito con il nome del controllo specifico:

```
ListorComboObject.List(ListorComboObject.ListIndex)
```

Il passo finale consiste nell'aggiungere il codice di DoStuff, in cui vengono effettivamente prese le decisioni di controllo di flusso. Ovviamente, in un'applicazione vera, dall'interno dell'istruzione Select. . .Case si salterà ad altre routine dal nome appropriato, invece di limitarsi a chiamare MsgBox (vedere la Figura 4.8).

Figura 4.8 *Questo utente oggipercolazione prende "Pigs in a Blanket".*

```
private Sub DoStuff(Which As String)
    Select Case Which
        Case "W"
            MsgBox "Oggi ha scelto i biscotti!"
        Case "F"
            MsgBox "Oggi hai scelto il toast!"
        Case "P"
            MsgBox "Oggi hai scelto il maiale!"
        Case "G"
            MsgBox "Il grana è servito!"
        Case "B"
            MsgBox "Che tipo di frutto di bosco desidera oggi, Signore?"
        Case Else
           MsgBox "Errore interno in DoStuff; chiamare lo sviluppatore!"
    End Select
End Sub
```

Fare attenzione a messaggi tipo "Incapace"

Nella procedura DoStuff, non c'è ragione perché venga eseguita la clausola Case Else. Nessuno mai vedrà la casella di messaggio "Internai Error", perché conosciamo le voci aggiunte alla casella di riepilogo, e sappiamo che abbiamo enumerato tutte le possibili scelte prima dell'istruzione Select. . .Case.
Tuttavia, è buona pratica di programmazione aggiungere un messaggio d'errore, proprio per il caso in cui avvenga qualcosa di bizzarro, come un errore di battitura nella digitazione del codice. I progetti della vita reale tendono a essere molto più complessi di questo esempio! Facendo così, se tale messaggio dovesse apparire, il debug del problema sarà un gioco da ragazzi. La gestione degli errori in VB viene trattata in dettaglio nel Capitolo 15.
I messaggi d'errore per situazioni ad hoc, come quello aggiunto alla procedura DoStuff, possono essere considerati come messaggi "poco gentili". Si racconta che uno sviluppatore abbia ricevuto una telefonata da un cliente a cui era apparso il messaggio "Errore per puntatore allocato male, incapace!". Fortunatamente, dice il racconto, il cliente aveva un buon senso dell'umorismo. Suppongo che la morale della favola del messaggio d'errore "Incapace" sia che ci si dovrebbe assicurare che tutti i messaggi d'errore diagnostici siano formulati in modo tale da non fare una brutta figura se a un cliente ne comparisse uno.

Un vantaggio che si ottiene passando un parametro a una struttura di decisione è che diventa estremamente facile aggiungere la stessa funzionalità ad altri gestori di eventi del programma, in quanto basta chiamare ancora DoStuff. Per esempio, l'utente dovrebbe essere in grado di effettuare la scelta nella finestra dell'esercizio precedente facendo doppio clic sulla casella di riepilogo lstSelect, senza dover fare clic sul pulsante cmdPass. Basta aggiungere il codice che chiama DoStuff al gestore dell'evento DblClick di lstSelect, come mostrato nel Listato 4.5.

Listato 4.5 *Richiamo di DoStuff da un evento di doppio clic.*

```
Private Sub lstSelect_DblClick()
    Dim Choice As String
    If      lstSelect.ListIndex     =  -1  Then
        'Controlla se è stato selezionato qualcosa
        MsgBox "Non è stato selezionato nulla!"
    Else
        Choice      =     lstSelect.List(lstSelect.ListIndex)
        'Prende la stringa selezionata
        DoStuff Left(UCase(Choice), 1)
    End If
End Sub
```

Un altro modo di fare la stessa cosa usando meno codice, e quindi forse preferibile, è chiamare una procedura di gestione di eventi dall'interno di un'altra. Questo è un modo facile di includere all'interno del primo gestore di eventi la funzionalità del secondo. Supponiamo di volere che l'evento Click del form si comForti come il gestore dell'evento Click di cmdPass, chiamando la struttura di decisione DoStuff. Si può semplicemente aggiungere una chiamata all'evento Click di cmdPass dall'evento Click di Form1:

```
Private Sub Form_Click()
    cmdPass_Click
End Sub
```

Le strutture di ciclo

Le strutture che iterano sono progettate per facilitare l'esecuzione ripetuta di una o più istruzioni. I cicli Do servono soprattutto quando non si sa precisamente quante volte debbano venire eseguite le istruzioni controllate, ma si conosce la condizione di uscita. I cicli For, d'altra parte, servono quando si sa quante volte debba venire eseguito un blocco di codice eseguibile. In generale, le strutture di ciclo possono venire annidate fra loro per quanti livelli si desidera. Comunque, ai fini della leggibilità del codice, sconsiglio di usare più di due livelli di annidamento. Se il flusso logico richiedesse livelli più profondi, si usino delle chiamate di subroutine, ponendo in tali subroutine le strutture dei livelli più profondi.
Qui vedremo una panoramica dei concetti implicati nelle strutture di ciclo, ma per la sintassi precisa consultate la guida in linea. Alcune questioni di ottimizzazione sofisticata dei cicli vengono trattate nella Parte III..

I cicli Do

Ci sono più modi per scrivere cicli Do in Visual Basic. Si può iterare fino a quando (Until) una condizione diventa vera (True) o fintanto (While) che una condizione rimane vera (True). In un ciclo Do Until o in un ciclo Do While, l'espressione di prova può venire posta all'inizio o alla fine della struttura di decisione. Quando l'espressione di test si trova all'inizio della struttura, viene valutata prima che le istruzioni della struttura vengano eseguite per la prima volta. Ciò significa che le istruzioni potrebbero non venire eseguite neanche una volta. D'altra parte, quando

il test condizionale si trova alla fine della struttura, si ha la garanzia che almeno un'esecuzione percorre le istruzioni della struttura.
Effettivamente, VB non ha un reale bisogno di avere entrambi i tipi di strutture Do perché, logicamente, Do Until è equivalente a Do While Not. Ancora una volta, Visual Basic lascia che ciascuno faccia a modo proprio!
Si può uscire dall'interno di una struttura Do partendo da qualunque punto del codice di esecuzione usando l'istruzione Exit Do, che fa saltare il flusso alla prima istruzione eseguibile che segue la struttura. Sebbene occasionalmente ci possano essere valide ragioni per usare un'uscita immediata da un ciclo, una migliore pratica di programmazione strutturata consiste nello stabilire le espressioni di prova in modo che non sia necessario usare quell'istruzione.

I cicli For

La struttura di ciclo For. . .Next include una variabile contatore. Usando questa struttura, si può controllare esattamente quante volte vengono eseguite le istruzioni della struttura. L'istruzione For di VB è molto flessibile, permettendo di impostare il valore iniziale del contatore, il suo valore finale e il suo incremento con qualunque valore numerico, positivo o negativo. Per specificare l'incremento si usa la parola chiave Step; se la si omette, l'incremento di default è 1.
Analogamente alla clausola Exit Do, anche Exit For provoca un salto immediato dell'esecuzione all'esterno della struttura di controllo.
Una variazione sul ciclo For è il ciclo For Each. . .Next, che ripete delle istruzioni di esecuzione per ogni elemento di un insieme di oggetti o di una matrice.

I moduli, le subroutine e le funzioni

Il codice sorgente dei progetti Visual Basic è suddiviso in moduli, che a loro volta sono composti di subroutine. Un sinonimo di subroutine è *procedura.* Un tipo imFortante di procedura è la *funzione,* che restituisce un valore.

I moduli

I progetti in codice sorgente Visual Basic sono costituiti da moduli, di cui esistono tre varietà: i moduli di form, i moduli standard e i moduli di classe.

I moduli di form

I moduli di form hanno .Frm come estensione del nome di file e contengono informazioni sui form che sono visibili agli utenti in fase di esecuzione. Possono contenere:

- Descrizioni grafiche di: proprietà dei form, controlli dei form, proprietà dei controlli dei form
- Dichiarazioni a livello di form
- Procedure generali
- Procedure di gestione di eventi

Per ulteriori informazioni sull'effettivo contenuto di un file .Frm, si può vedere il Capitolo 19.
I moduli di form sono di due varietà, a seconda delle caratteristiche dell'interfaccia utente del form: *iforni con interfaccia utente a singolo documento (Single Document Interface,* o SDI), e *i form con interfaccia utente a documenti multipli (Multiple Document Interface,* o MDI). Quando si usa il termine "form" da solo, in generale ci si riferisce ai form SDI. Si noti anche che le applicazioni MDI possono avere un solo form MDI. Le applicazioni MDI vengono trattate nella Parte IV.
Per aggiungere un nuovo modulo di form a un progetto, scegliere *Form* o *MDI Form* dal menu *Insert.*

I moduli standard

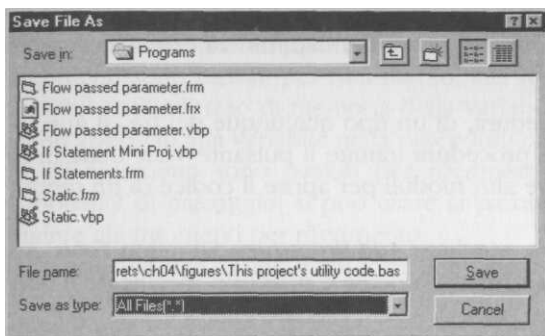
I moduli standard hanno .Bas come estensione del nome di file. I file .Bas contengono librerie di codice sorgente. Possono includere dichiarazioni globali o a livello di modulo, e procedure globali. Per aggiungere un modulo .Bas, scegliere *Add Module* dal menu *Project.*

I moduli di classe

I moduli di classe, che hanno .Cls come estensione del nome di file, servono a creare nuovi oggetti. I moduli di classe sono moduli standard che possono contenere proprietà, metodi ed eventi. Si può usare la parola chiave *New* per creare un nuovo esemplare di un oggetto basato su un modulo di classe e ottenere un puntatore a tale esemplare. La sezione "Parlare il linguaggio degli oggetti" più avanti in questo capitolo tratta alcuni dei meccanismi di sintassi riguardanti le classi; il Capitolo 14, tratta in dettaglio la programmazione orientata agli oggetti e la creazione dei moduli di classe.
Per aggiungere un modulo di classe a un progetto, bisogna scegliere *Add Class Module* dal menu *Project.*

Nomi di file lunghi in progetti VB6 sotto Windows a 32 bit

Nelle versioni a 32 bit di Windows (per esempio 98, 95 e NT) si possono usare nomi di file lunghi che incorForano spazi nei nomi del progetto e del modulo. Questo segna la fine una volta per tutte di quei "non ho la più pallida idea di cosa faccia questo modulo" dovuti a nomi di file indecifrabili (vedere Figura 4.9).
Sebbene sia positivo poter attribuire ai moduli nomi comprensibili, affinchè sia chiaro cosa contengono, esistono anche valide ragioni per restare fedeli al vecchio standard 8+3 del DOS. Per esempio, lo standard ISO9660 per i CD-ROM non supForta i nomi lunghi per i file.

Figura 4.9
in VB6 si possono usare i nomi di file lunghi For i moduli.

Procedure

Come si può dedurre dalla descrizione di ogni tipo di modulo, in generale, le Forzioni di codice dei tre tipi di moduli sorgente sono costituite da dichiarazioni, di cui abbiamo già parlato all'inizio del capitolo, e da procedure. Esistono tre tipi di procedure: Sub, Function e Property:

- Le procedure Sub sono subroutine. Il codice all'interno di una procedura Sub sarà eseguito quando la Sub sarà chiamata. Le procedure Sub non hanno un valore di ritorno.

  Le procedure Sub opzionalmente possono essere dichiarate mediante le parole chiave Private/Public. (L'assenza della parola chiave equivale all'uso di Public.) Private significa che la procedura Sub è visibile solo nel suo modulo; Public significa che è visibile globalmente in un progetto.

  L'uso opzionale della parola chiave Static significa che le variabili locali della procedura Sub sono preservate tra una chiamata e l'altra alla procedura.

  Le procedure Event sono un tipo particolare di procedura Sub. Vengono usate per memorizzare una procedura di gestione degli eventi (vedere Capitolo 3). Le procedure Event sono sempre procedure Sub dichiarate Private e con un underscore (_) che separa un oggetto e il suo evento, per esempio:

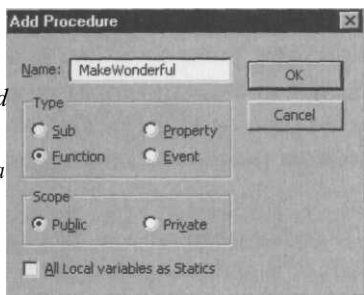
  ```
  Private Sub  Form1_Click  ()
  ```

- Le procedure Function sono come le procedure Sub, tranne per il fatto che restituiscono un valore. (Si osservi che VB fornisce numerose funzioni di sistema: non dovete scrivervele, dovete solo chiamarle, usarle ed essere contenti della loro presenza. Alcune di queste funzioni verranno trattate successivamente in questo capitolo.)

  Tenete presente che, per quanto riguarda la terminologia, le parole "procedura" e "routine" vengono generalmente usate per fare riferimento a subroutine o a funzioni. Dovete pensare a una funzione semplicemente come a un tipo di procedura, una procedura che restituisce un valore.

- Le procedure *Property* sono usate per creare e manipolare le proprietà di forni e altri moduli. Ne parleremo nel Capitolo 14 e anche discorsivamente in altri punti del libro.

Per aggiungere una procedura, di un tipo qualunque dei tre, si apre il modulo nel quale si vuole inserire la procedura tramite il pulsante *View Code* in *Project Explorer*. (Il Capitolo 3 descrive altri moduli per aprire il codice di un modulo nella finestra *Code.)*
Con la finestra *Code* attiva, scegliere *Add Procedure* dal menu *Tools*. Appare la finestra di dialogo *Add Procedure,* nella quale è possibile specificare il nome e le caratteristiche della procedura che si desidera (vedere Figura 4.10).

Figura 4.10 *Si può usare la finestra di dialogo Add Procedure per creare una meravigliosa funzione pubblica!*

Duello che segue è il codice del modello creato in base alle selezioni fatte nella Figura 4.10:

```
Public Function MakeWonderful ()

End Function
```

Adesso si può inserire una lista di argomenti per la funzione appena creata digitandola tra parentesi. (A questo punto è anche possibile assegnare un tipo esplicito alla funzione.)

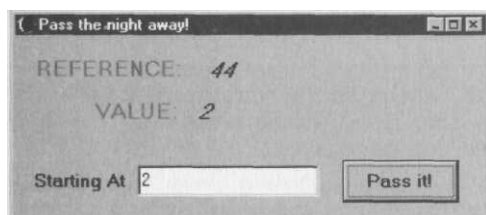
```
Public Function MakeWonderful (Orfeo As String,
   Euridice As Variant) As Integer

End Function
```

La lista di argomenti per una procedura o funzione è qualcosa alla quale si fa riferimento con il termine *parametri formali,* distinguendo così la lista da quella fornita Quando la procedura viene chiamata, la quale include i *parametri effettivi.*

Passaggio di argomenti

I valori che vengono passati alle procedure sono noti come *argomenti o parametri.* Gli argomenti possono essere passati per *valore* o per *riferimento.*
Quando gli argomenti sono passati per valore, viene inviata alla procedura solo una copia della variabile passata. In questo caso eventuali cambiamenti aportati alla

variabile nella procedura chiamata non interessano l'originale. Si usa la parola chiave ByVal per specificare che un parametro viene passato per valore.
Quando gli argomenti sono passati per riferimento, alla procedura chiamata viene passato un puntatore all'indirizzo di memoria della variabile. In questo caso eventuali cambiamenti apFortati alla variabile nella procedura chiamata interessano l'originale. In VB gli argomenti sono passati per riferimento di default, se non si specifica una modalità di passaggio; si può usare la parola chiave ByRef per indicare esplicitamente gli argomenti per riferimento.
Si tenga presente che una lista di parametri formali di una procedura può tranquillamente includere un mix di parametri per riferimento e per valore. In altre parole, i parametri di una routine non devono essere necessariamente uguali sotto questo aspetto. (La frase "lista di parametri formali" si riferisce alla lista di parametri dichiarati nell'intestazione di una procedura e si contrappone alla lista di variabili effettive passate nella chiamata alla procedura.)
Per mostrare la differenza tra passaggio per riferimento e per valore, creerò due procedure molto semplici (un'istruzione che somma 42 all'argomento passato) che differiscono tra loro per un solo aspetto: DemoRef accetta i suoi argomenti per riferimento, mentre DemoVal li accetta per valore. DemoRef e DemoVal saranno chiamate con variabili che sono state impostate in modo identico usando un'istruzione di assegnamento prima di chiamare le routine. Come si può constatare DemoRef ha aggiunto 42 alla variabile nella procedura chiamante, invece DemoVal no (vedere Figura 4.11).

Figura 4.11
Il passaggio di parametri per riferimento o per valore può condurre a risultati diversi.

L'uso delle variabili varianti provoca le conversioni di tipo appropriate

Nell'esempio del passaggio di parametri, l'uso di variabili dichiarate come Variant fa sì che abbiano luogo automaticamente le conversioni corrette dal contenuto di tipo stringa della casella di testo txtStart a un valore numerico, e viceversa da un numero a una didascalia di etichetta di tipo stringa. Se avessimo dichiarato la variabile come, per esempio, intera (Integer), avremmo dovuto usare la funzione Val, che converte le stringhe in numeri, per ottenere l'input iniziale, e poi la funzione Str, che con verte i numeri in stringhe, per visualizzare il risultato.

Per preparare il progetto, aggiungere due etichette, lblRef e lblValue, al suo form per visualizzare i risultati della dimostrazione. Poi, aggiungere una casella di testo, txtStart, in cui l'utente può introdurre un valore iniziale per la chiamata di subroutine, e un pulsante di comando, cmdPass, per avviare la dimostrazione. Si noti che non ci preoccupiamo della *convalida dell'input.* Non c'è nessuna verifica che l'utente abbia effettivamente introdotto un numero in txtStart. In un programma reale, si scriverà del codice di convalida dell'input per assicurarsi che gli utenti non possano inserire valori non leciti.

Ecco le sub DemoRef e DemoVal:

```
Private Sub DemoRef(Argument)  'Passato per riferimento
    Argument = Argument + 42
End Sub

Private Sub DemoVal(ByVal Argument)  'Passato per valore
    Argument = Argument + 42
End Sub
```

Il Listato 4.6 mostra il codice cmdPass_Click che esegue la demo:

Listato 4.6 *Esecuzione della demo.*

```
Private Sub cmdPass_Click()
    Dim ArguRef, ArguVal
    ArguRef = txtStart
    ArguVal = ArguRef       'Sono uguali
    DemoRef ArguRef
    DemoVal ArguVal
             'Ora non lo sono, anche se le routine DemoRef
             'e DemoVal sono identiche, tranne che
             'per la modalità dei parametri!
    lblRef = ArguRef
    lblValue = ArguVal
End Sub
```

Dopo di ciò, penso che sia davvero difficile non avere le idee chiare sulla differenza fra passaggio per riferimento e passaggio per valore!

Ecco alcune altre imFortanti tecniche di passaggio di parametri:

- Utilizzo di argomenti opzionali. Se si usa la parola chiave Optional in un elenco di parametri, non è obbligatorio passare i successivi parametri formali, ma lo si può fare se lo si desidera.
- Utilizzo di un numero di argomenti indefinito. La parola chiave ParamArray in un elenco di parametri formali permette di specificare che la procedura accetterà un numero indefinito di argomenti.
- Si può usare l'operatore di equivalenza, rappresentato dal segno di due-punti-uguale (:=) per passare e accettare parametri secondo il nome formale del parametro, indipendentemente dal suo ordine nell'elenco di parametri.

Le strutture definite dal programmatore

Le strutture definite dal programmatore, chiamate anche *tipi definiti dall'utente,* permettono di creare i propri tipi di dati come combinazione di tipi di variabili esistenti. I tipi definiti dall'utente possono chiarificare grandemente la logica del programma. È difficile immaginare un programma ben scritto di qualunque complessità che non ne faccia alcun uso.

Un tipo definito dall'utente viene definito nella sezione Declarations di un modulo usando la parola chiave Public o Private, seguita dalla parola chiave Type (tipo), dal nome della struttura definita dall'utente, da un elenco delle variabili che compongono il tipo, una per riga, e infine dalle parole chiave End Type. Per usare una variabile basata su un tipo definito dall'utente, si deve fare ancora un altro passo: dichiarare la variabile, al solito modo, come basata su tale tipo.

Ecco un esempio molto semplice di un tipo definito dall'utente e di alcune dichiarazioni di variabile basate su di esso:

```
Option Explicit

'Sezione delle dichiarazioni di Form1
Private Type Employee
    FullName As String
    SSN As Long
    Rating As Integer
    DOB As Date
End Type
Private JacksonW, HopperB, EggertM As Employee
```

Tutto tranne il lavandino della cucina ...

In un tipo definito dall'utente possono andare tutti i tipi di cose, compresi variabili varianti, matrici e oggetti. Tali inclusioni possono creare strutture molto flessibili e potenti. Per esempio, un elemento potrebbe essere un esemplare di un form:

```
Private Type RolePlay
    UserInput As Form
    dbUserList As Database
End Type
```

Comunque, esiste come contropartita il consumo di risorse, particolarmente quando si definisce una matrice di varianti come parte di una struttura. Tratteremo in seguito queste questioni e le relative tecniche di programmazione.

Per immagazzinare e recuperare i valori dagli elementi di una struttura definita dall'utente si usa l'operatore punto (.), proprio come quando si accede alle proprietà di un oggetto. Per esempio:

```
Dim HopperB As Employee
HopperB.Rating = 99
If HopperB.Rating > 80 Then MsgBox "Ottimo!"
```

Con Visual Basic 6, gli argomenti e i tipi restituiti da proprietà e da metodi pubblici possono essere tipi definiti dall'utente.

Le matrici

Come molti sanno, una *matrice* (array) si riferisce a una serie di variabili con lo stesso nome che usano uno o più indici. Le matrici possono essere monodimensionali o multidimensionali. Possono, esse stesse, essere popolate da matrici, se sono di tipo Variant. Possono includere strutture definite dall'utente e possono esservi incluse.
Nel seguito del libro, vedremo come operare con le matrici di controlli e con le matrici di form, come anche con gli insiemi di oggetti. In effetti, e continuerò a battere su questo punto finché sarò certo che sia veramente assimilato, le variabili oggetto *in realtà* sono puntatori a oggetti.

La differenza fondamentale fra le matrici e gli insiemi sta nel fatto che il numero indice di una matrice può venire usato per far riferimento a elementi specifici di una matrice.

Chi fosse interessato a ulteriori informazioni sulle operazioni con le matrici e gli insiemi di oggetti, passi ai Capitoli 13 e 14.
La sintassi della dichiarazione e della manipolazione delle matrici VB è flessibile e anche facilmente comprensibile. Suggerirei di dare un'occhiata alla sezione sulle matrici nella documentazione del prodotto. Anche in questo libro, in seguito, verranno trattati alcuni degli aspetti più sottili della gestione delle matrici e dell'ottimizzazione delle prestazioni, particolarmente nella Parte III.
Un aspetto della gestione delle matrici in VB è così utile e facile da usare, perfino per la meravigliosa piattaforma di sviluppo che è Visual Basic, che vale la pena di sottolinearlo. VB permette di creare matrici che sono *dinamiche* in fase di esecuzione, nel senso che si può cambiare la dimensione delle matrici, eventualmente a seconda dell'input dell'utente, mentre il programma viene eseguito. Lo si fa dichiarando originariamente la matrice con un elenco di dimensioni vuoto e usando la parola chiave `ReDim` quando si vuole allocare l'effettivo numero di elementi, sia la prima volta che le successive.

Preservare il contenuto delle matrici dinamiche

Se si usa ReDim per ridimensionare una matrice in fase di esecuzione, tutti i valori correnti nella matrice vengono azzerati. Se si vuole modificare la dimensione di una matrice mantenendo il valore corrente degli elementi, si deve usare la parola chiave Preserve nell'istruzione ReDim. Per esempio.

```
Dim TheArrayO As Integer   ' Dichiara una matrice dinamica
Redim TheArray(5)          ' Alloca 5 elementi.
For I = 1 To 5             ' Cicla cinque volte.
  TheArray(I) = I          ' Inizializza la matrice.
Next I
Redim TheArray(10)         ' Ridimensiona a 10 elementi, cancella
                           ' i valori di tutti gli elementi
For I = 1 To 10            ' Cicla dieci volte
  TheArray(I) = I          ' Inizializza la matrice
Next I
Redim Preserve TheArray(15)  ' Ridimensiona a 15 elementi,
                             ' conservando i valori
                             ' degli elementi esistenti
```

In VB6, le funzioni e le procedure di proprietà adesso possono restituire matrici. Inoltre, le matrici a dimensione variabile adesso possono apparire sul lato sinistro di un 'istruzione di assegnamento.

Parlare il linguaggio degli oggetti

Oggetti, oggetti, oggetti! Tutto, o quasi, è un oggetto.
Si possono usare gli oggetti per estendere la potenza dell'ambiente VB e per strutturare le proprie applicazioni; inoltre, si può usare VB per creare degli oggetti che altri possono cogliere dall'ampio oceano di componenti ActiveX basati su OLE per usarli nelle loro applicazioni o sul Web.
Tutte le applicazioni, tranne le più semplici e meno sofisticate, implicheranno qualche interazione con degli oggetti. Ogni programmatore VB può pensare che le proprie interazioni con gli oggetti cadano nelle seguenti categorie generali:

- Utilizzo di controlli ActiveX nelle proprie applicazioni
- Utilizzo di componenti ActiveX, cioè oggetti server OLE, nelle proprie applicazioni
- Creazione e utilizzo interno di oggetti, come Form e oggetti basati su classi
- Creazione di componenti ActiveX, cioè applicazioni server OLE, utili a sé o ad altri
- Creazione di controlli e documenti ActiveX per il proprio uso, da far usare ad altri come componenti, o per l'uso sul Web

Questa sezione fornisce informazioni sulla sintassi basilare per operare con i controlli ActiveX, con i componenti ActiveX, con i server OLE e con altre librerie. Per ulteriori informazioni, si dovrebbe dare un'occhiata ai Capitoli 14 e 23. Nel Capitolo 29, si troveranno anche informazioni su come creare un tipo molto speciale di componente ActiveX, che manipola esemplari dell'ambiente Visual Basic stesso.
La creazione dei controlli ActiveX è trattata in dettaglio nella Parte VI. Nel Capitolo 28 si troveranno informazioni sui documenti ActiveX.

Utilizzo dei controlli ActiveX

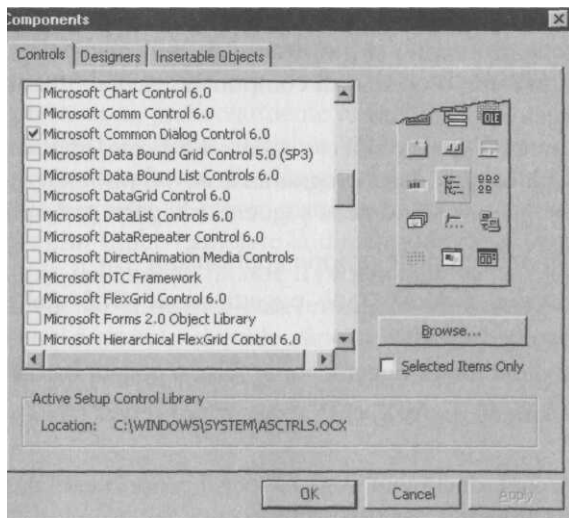
Per usare un controllo ActiveX, si deve aggiungere il controllo alla Toolbox usando la finestra di dialogo *Components,* a cui si accede dal menu *Project,* come mostrato nella Figura 4.12. Dopo aver aggiunto un controllo alla Toolbox, si può fare doppio clic sul controllo per aggiungerne un esemplare a un form. Quando l'esemplare del controllo è stato aggiunto al form, si può far riferimento alle sue proprietà e ai suoi metodi nel codice.

L'Object Browser è uno strumento eccellenteper trovare leproprietà e i metodi disponibili di un esemplare di controllo che è stato inserito.

Gli assegnamenti di proprietà possono venire fatti usando l'operatore punto. Per esempio, se `myTooll` è il nome di un controllo avente la proprietà `Caption`:

```
Dim OldCaption as String
OldCaption = Form1.myTool1.caption
Form1.myTooll.caption = "Frodo"
```

Si noti che all'interno di un dato modulo, come Form1 nell'esempio, non è necessario usare il nome del modulo per invocare una proprietà di un controllo:

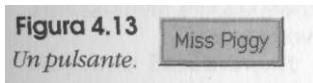
Figura 4.12 *La scheda Controls della finestra di dialogo Components serve ad aggiungere controlli ActiveX al proprio progetto.*

```
myTool1.caption = "Frodo"
```

funzionerebbe altrettanto bene, purché si introduca tale codice all'interno del modulo che fa da contenitore per tale controllo.
Si può usare l'istruzione With...End With per scrivere del codice più pulito che coinvolge degli oggetti, attraverso un riferimento implicito esteso all'oggetto. Per esempio, se su un form si ha un pulsante di comando di nome cmdDemo, collocando il seguente codice nell'evento Load del form viene usato un riferimento implicito per modificare le proprietà del pulsante, così che l'espressione

```
cmdDemo.caption
```

e le altre espressioni analoghe non devono venire formulate esplicitamente. Questo diventa di aiuto ancora maggiore quando si hanno degli oggetti annidati dentro gli oggetti che si sta manipolando. Ecco il codice d'esempio, con i risultati mostrati in Figura 4.13:

```
Private Sub Form_Load()
    With cmdDemo
        .Caption = "Miss Piggy"
        .Font.Size = 12
        .Height = 620
        .Width = 1400
        .Default = True
    End With
End Sub
```

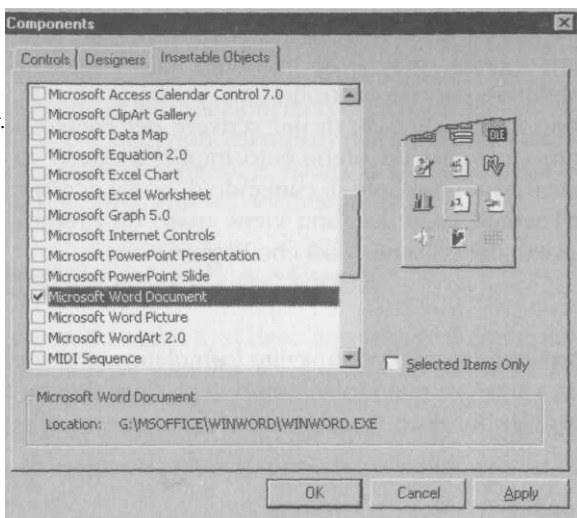
Figura 4.13 *Un pulsante.*

I metodi sono implementati internamente ai controlli ActiveX come funzioni. Vengono chiamati allo stesso modo: come funzioni con argomenti. Per esempio, il controllo myTool potrebbe avere il metodo DoSomething con un argomento di tipo stringa. Se un esemplare di myTool fosse collocato su Form1, si invocherebbe il metodo come segue:

```
Form1.myTool1.DoSomething("Prendiquestastringa!")
```

I controlli ActiveX aggiunti a un contenitore, come un form, possono far scattare degli eventi. Non si devono confondere questi eventi con quelli *ricevuti* dai controlli. Si risponde agli eventi dei controlli nello stesso modo in cui si risponde agli eventi dei form: collocando del codice che gestisce l'evento nell'intelaiatura creata da Visual Basic per i programmatori.

Utilizzo dei componenti ActiveX

Molti componenti ActiveX, cioè le applicazioni server OLE, forniscono oggetti che possono venire inseriti in un contenitore OLE. La scheda *Insertable Objects* della finestra di dialogo *Components,* aperta dal menu *Project,* serve a questo scopo, come mostrato in Figura 4.14. Per esempio, si potrebbe aggiungere un documento Word.

Figura 4.14 *Si possono aggiungere oggetti inseribili OLE alla Toolbox.*

Quando si fa doppio clic sull'icona dell'oggetto inseribile nella Toolbox e lo si aggiunge a un contenitore, normalmente vengono visualizzate le toolbar del server OLE; nell'esempio quelle di Microsoft Word.
Si può accedere alle proprietà, agli eventi, e ai metodi dell'oggetto OLE inserito proprio come si farebbe con quelli di un controllo ActiveX. Per esempio, si potrebbe visualizzare un messaggio all'utente quando questo ha finito di modificare un documento Word inserito, collocando del codice nell'evento LostFocus dell'oggetto; questo evento scatta quando un oggetto perde lo stato attivo (focus):

```
Private Sub Document1_LostFocus()
   MsgBox "Sei sicuro di avere finito?"
End Sub
```

Oltre agli oggetti OLE che forniscono oggetti con interfacce visibili da aggiungere visivamente ai contenitori, si può creare un esemplare di qualunque oggetto OLE e usarlo nel proprio codice. Si può creare un esemplare di un oggetto OLE, noto anche come *oggetto ActiveX,* usando la funzione CreateObject. Per esempio:

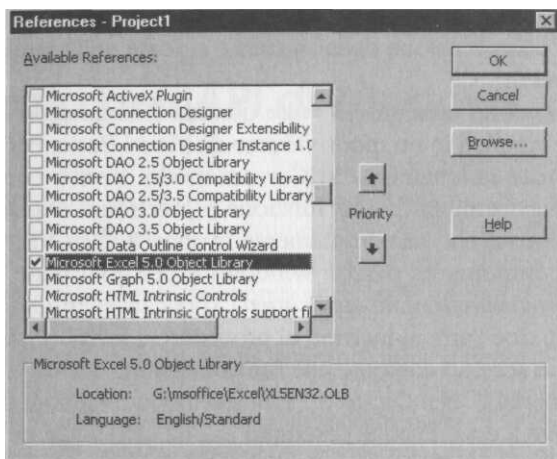
```
Dim ExcelSheet As Object
Set ExcelSheet = CreateObject("Excel.Sheet")
```

La variabile ExcelSheet adesso tiene un esemplare di un oggetto Excel.Sheet. Si parla, in casi come questo, di *associazione tardiva (late binding),* perché il compilatore VB non sa che tipo di oggetto andrà nella variabile oggetto finché non è effettivamente assegnata.
Le proprietà e i metodi esposti di questo oggetto, chiamati anche *membri,* si possono manipolare nel codice come si preferisce.
È imFortante alla fine liberare la memoria riservata per gli oggetti che vengono creati. Questo avviene automaticamente quando la variabile che immagazzina il riferimento all'oggetto esce dall'ambito d'azione; per esempio, una variabile a livello di

procedura esce dall'ambito d'azione quando l'esecuzione lascia la procedura. Lo si può anche fare esplicitamente, usando la parola chiave Nothing:

```
Set ExcelSheet = Nothing
```

Usando la finestra di dialogo *References,* che si apre dal menu *Project,* si può aggiungere un riferimento a una libreria di oggetti (vedere la Figura 4.15).

Figura 4.15 *La finestra di dialogo References serve ad aggiungere al proprio progetto un riferimento a una libreria di oggetti.*

Senza ulteriore fatica, si possono usare nel codice gli oggetti che fanno parte delle libreria a cui è stato stabilito il riferimento:

```
Dim X As Excel.Sheet
```

Chiamata di procedure esterne

Uno dei modi più imFortanti di iniziare a estendere la potenza di VB oltre le sue impressionanti funzionalità di base, è tramite la chiamata a procedure e funzioni che si trovano in librerie compilate esterne. Generalmente, queste sono *librerie a collegamento dinamico* (Dynamic Link Library), indicate come DLL, ma una libreria compilata non ha bisogno di avere nessuna particolare estensione al nome di file. Per esempio, una libreria a collegamento dinamico potrebbe venire salvata con le estensioni .Exe, .Cpl, o .Scr, per dirne alcune.

È molto facile usare le procedure esterne. Questi sono i due passi:

1. Si usa l'istruzione Declare, nella sezione General Declarations di un modulo, per identificare la libreria esterna, la procedura, i parametri della procedura e l'ambito d'azione dei riferimenti alla procedura esterna. Le Sub o le funzioni esterne che vengono dichiarate con la parola chiave Private sono disponibili entro il modulo della dichiarazione; quelle dichiarate Public sono disponibili a un intero progetto.

2. Usando dei parametri appropriatamente tipizzati, si chiama la procedura normalmente dall'interno del proprio codice in qualunque posto a cui si estenda l'ambito d'azione dell'istruzione Declare. Usare la procedura esterna, dopo che è stata dichiarata, è esattamente lo stesso che usare una procedura o funzione scritta in Visual Basic.

Sebbene l'effettiva sintassi della dichiarazione esterna non sia difficile, è un po' complessa a causa di tutte le diverse possibilità implicate. In seguito vedremo moltissimi esempi di dichiarazioni esterne; per adesso, la cosa migliore che potete fare per avere una completa comprensione della sintassi è cercare nella guida in linea di V*B* sotto "Declare Statement".
Alcune altre questioni possono emergere a volte quando il linguaggio della procedura esterna definisce le variabili in un modo diverso da Visual Basic. Per esempio, le stringhe sono rappresentate internamente in C in un modo diverso che in VB. Ciò significa che i valori stringa restituiti da una funzione o procedura esterna C devono venire convcrtiti in un formato che sia correttamente riconosciuto da VB. Questa questione è particolarmente imFortante, perché Windows è per gran parte scritto in C. Perciò, *l'interfaccia di programmazione applicativa* (Application Programming Interface, o API) di Windows, cioè l'ampio insieme di procedure e funzioni progettate per dare ai programmatori un accesso coerente alle funzionalità interne di Windows, usa a tipizzazione di variabili del C. Nel Capitolo 11 tratteremo in maggiore dettaglio tale questione e quelle relative a essa. Ecco un esempio di una dichiarazione di funzione esterna che verrebbe collocata nella sezione General Declarations di Form1 :

```
Option Explicit
Private Declare Function Hobbit Lib "Mydll"
    Alias "#1" (Which As Integer) As String
```

Questo indicherebbe che la funzione di nome Hobbit è stata dichiarata così da poter venire usata solamente entro il modulo Form1. La libreria che contiene la funzione Hobbit ha nome Mydll.Oli; se l'estensione del nome di file fosse stata diversa da .Dll, avrebbe dovuto essere specificata. La funzione Hobbit è stata fornita da Mydll con un numero indice ordinale, noto anche come punto d'ingresso *(entry point)*, come specificato nella clausola Alias dell'istruzione Declare. La funzione Hobbit accetta come parametro passato per riferimento un valore intero e restituisce una stringa. Ci si dovrebbe anche rendere conto che una libreria esterna non può venire caricata se non viene trovata. La cosa migliore da fare è solitamente collocare le DLL nella directory di esecuzione. Alternativamente, le si può mettere nella directory Windows\ System, in qualunque directory a cui fa riferimento la variabile d'ambiente PATH, o dichiararla con un percorso esplicito:

```
Private Declare Function Hobbit Lib _
    "C:\VbSecrets\Program\Mydll.Dll"
```

Un esempio nel Capitolo 16 spiega come preparare un progetto VB in modo che cerchi un file, come una DLL, che non si trova nella posizione giusta. Nel progetto d'esempio, vedremo come il software, se non riesce a trovare il file giusto, può dare all'utente la possibilità di trovarlo e di rilanciare il progetto.

Ecco come la funzione Hobbit potrebbe venire chiamata dall'interno del gestore dell'evento Click di Form1:

```
Private Sub Form_Click()
    MsgBox Hobbit(1)
End Sub
```

Quando l'evento scatta, la funzione MsgBox dovrebbe visualizzare il valore di stringa reso da Hobbit quando gli viene passato il parametro 1.

Chiamata dell'API di Windows

Chiamare una delle procedure o funzioni che fanno parte dell'API di Windows implica esattamente gli stessi passi fatti per chiamare ogni altra procedura esterna. Dapprima, si deve dichiarare formalmente la procedura esterna a livello di modulo. Poi, si usa la procedura con un appropriato elenco di argomenti. Fortunatamente, Visual Studio fornisce uno strumento che rende un gioco da ragazzi aggiungere le dichiarazioni per l'API.
Le procedure e funzioni che costituiscono il nucleo dell'API di Windows si trovano in tre file di libreria, che hanno estensione .Dll. La Tabella 4.3 elenca i file dell'API di Windows a 32 bit, insieme ai corrispondenti nomi di libreria dell'API di Windows 3.x a 16 bit per riferimento storico.

Tabella 4.3 *File delle librerìe API a 32 bit e a 16 bit.*

Windows 98,95, e NT	Windows 3.x
User32.Dll	User.Exe
Gdi32.Dll	Gdi.Exe
Kernel32.Dll	Krnl386.Exe

Per facilitare l'aggiunta di dichiarazioni API, Microsoft ha incluso un'applicazione in Visual Studio Professional Edition, l'API Text Viewer (visualizzatore di testo dell'API). Per aprire l'API Text Viewer, sia usa il menu *Start* di Windows per trovare la voce di programma Microsoft Visual Studio 6.0 Tools. Il Viewer si trova nel livello di menu successivo.

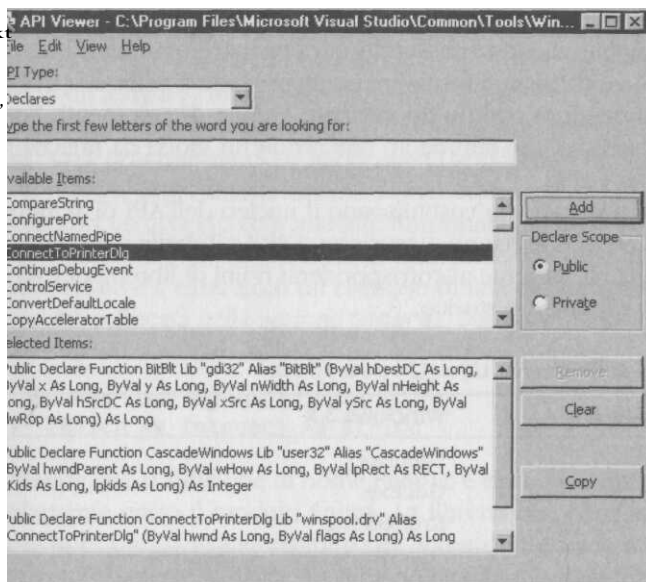
Visual Basic è dotato di un'aggiunta che abilita ad accedere all'API Text Viewer dall'interno dell'ambiente VB. Per ulteriori informazioni, si veda il Capitolo 29.

La prima volta che si apre l'API Text Viewer, lo si deve caricare con un file di dati sull'API. Qui si può scegliere se impostare l'API Viewer in modo che funzioni con un file di testo normale o con un file di database Access .Mdb. La contropartita è il tempo di configurazione iniziale, perché l'opzione database può richiedere un bel po' per prepararsi la prima volta. Naturalmente, per gli usi successivi l'opzione database è parecchio più veloce.
Dopo che l'applicazione è stata preparata, si può selezionare tra *Constants, Declares,* o *Types.* Il procedimento prosegue aggiungendo alla finestra *Selected Items* le

voci che si vorranno, e poi usando il pulsante *Copy* per metterle negli Appunti di Windows. In Figura 4.16, sono state selezionate le dichiarazioni per le routine dell'API BitBlt, CascadeWindows, e ConnectToPrinterDlg.

Se si fa clic sul pulsante *Copy,* le dichiarazioni per queste procedure vengono copiate negli Appunti. Il prossimo passo consiste nell'andare alla sezione Declarations di un modulo del proprio progetto VB e incollare, usando il comando *Paste* del menu *Edit,* oppure le combinazioni di tasti dell'interfaccia CUA (Common User Access).

Il Listato 4.7 mostra le dichiarazioni risultanti, pronte da usare, come appaiono dopo averle incollate. Le righe sono state spezzate aggiungendo dei caratteri di continuazione di riga.

Figura 4.16 *L'API Text Viewer serve a copiare le dichiarazioni, costanti i tipi e le dell'API*

Listato 4.7 *Dichiarazioni dell'API incollate dall'API Text Viewer.*

```
Option Explicit
Declare Function BitBlt Lib "gdi32"
   (ByVal hDestDC As Long, ByVal x As Long,
   ByVal y As Long, ByVal nWidth As Long, _
   ByVal nHeight As Long, ByVal hSrcDC As Long,
   ByVal xSrc As Long, ByVal ySrc As Long,
   ByVal dwRop As Long) As Long
Declare Function CascadeWindows Lib "user32"
   (ByVal hwndParent As Long, ByVal wHow As Long,
   ByVal IpRect As RECT, ByVal cKids As Long,
   Ipkids As Long) As Integer
Declare Function ConnectToPrinterDlg Lib "winspool.drv1
   (ByVal hwnd As Long, ByVal flags As Long) As Long
```

Se provate a eseguire un progetto in cui abbiate incollato queste particolari dichiarazioni, otterrete un messaggio d'errore basato su un tipo non definito: "User-defined type not recognized". Questo è dovuto al fatto che RECT è un tipo definito da Windows, che specifica un rettangolo, che deve venire definito, se si intende usare procedure che usano tale tipo.
Si potrebbe cercare la corretta definizione del tipo, ma è più facile tornare all'API Text Viewer. Questa volta, si selezioni *Types* nella casella di riepilogo a discesa *API Type.* Successivamente, si scorra in giù finché si trova *RECT.* Lo si aggiunga alla casella di riepilogo *Selected Items,* lo si copi, e si incolli la definizione nel progetto:

```
Type RECT
        Left As Long
        Top As Long
        Right As Long
        Bottoni As Long
End Type
```

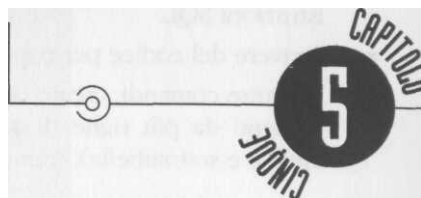
Questo è veramente tutto ciò che serve per aggiungere le dichiarazioni di API con l'API Text Viewer. Ripeto, se non fosse una perdita di tempo, niente impedirebbe di cercare l'appropriata sintassi delle dichiarazioni e di introdurle a mano nel proprio progetto VB.
Un ultimo modo di semplificare la chiamata delle API nei propri progetti consiste nel creare in Visual Basic un server OLE out-of-process che incapsuli le chiamate API di Windows, semplificando così i programmi che usano l'API. Non c'è molto da dire su questo processo, una volta che si siano compresi i fondamenti della tecnica OLE. Si troverà una dimostrazione nel Capitolo 10.

Riepilogo

Sebbene non sia una definizione formale del linguaggio, questo capitolo ha trattato il linguaggio Visual Basic dal punto di vista del programmatore esperto. Ci si è rivolti a lettori che avessero già assimilato i concetti di base, esaminati qui solo di sfuggita. Ci si è concentrati sul meraviglioso, intuitivo, e potente linguaggio Visual Basic 6. In particolare, ho cercato di spiegare gli elementi di linguaggio che servono a creare programmi professionali. Lungo la strada, sono stati illustrati alcuni concetti linguistici con esempi di codice che potete usare direttamente nei vostri progetti.

- È stato dimostrato come spezzare su più righe una singola stringa di casella di messaggio.
- Sono state mostrate delle tecniche per aggiungere intestazioni di commento ai moduli e alle routine, e per salvarle per poterle riutilizzare.
- Abbiamo visto come trovare il valore di stringa della voce selezionata in un controllo casella di riepilogo.
- È stato dimostrato come effettuare diramazioni in modo modulare a seconda della voce selezionata in una casella di riepilogo.
- Abbiamo visto come usare le proprietà di default dei controlli.

- È stata spiegata la differenza fra passare parametri per riferimento e per valore.
- Abbiamo visto come usare la sintassi necessaria per operare con gli oggetti.
- Abbiamo visto come dichiarare e chiamare una procedura che si trova in una libreria esterna.
- Abbiamo visto come dichiarare e chiamare l'API di Windows.

CARATTERISTICHE DI LIVELLO AVANZATO

- L'ambiente Data
- Il Data Object Wizard
- Controlli persistenti su pagine di Internet Explorer
- L'evento Validate dei controlli
- Aggiunta dinamica di controlli
- Restituzione di una matrice da una funzione
- Il modello ad appartamento di multithreading
- La funzione CallByName
- Nuove funzioni di stringa

Visual Basic 6 rappresenta un avanzamento incrementale dal Service Pack 3 di Visual Basic 5. Molti bachi sono stati corretti, e Visual Basic è ormai strettamente integrato con Visual Studio 98. Al nucleo di VB5 sono stati aggiunti degli strumenti di sviluppo Enterprise. Inoltre, sono stati aggiunti al prodotto molti nuovi strumenti imFortanti e caratteristiche di livello avanzato. Chi è pratico di VB5, non avrà problemi nell'uso di VB6. Tuttavia, a meno di sapere dove guardare, si rischia di non sfruttare le sue nuove caratteristiche.
Questo capitolo dà un assaggio di molte delle nuove affascinanti caratteristiche di livello avanzato di Visual Basic 6. Il capitolo fornisce semplicemente una panoramica, ma dice anche dove saltare all'interno del libro per avere informazioni più dettagliate sugli argomenti trattati.

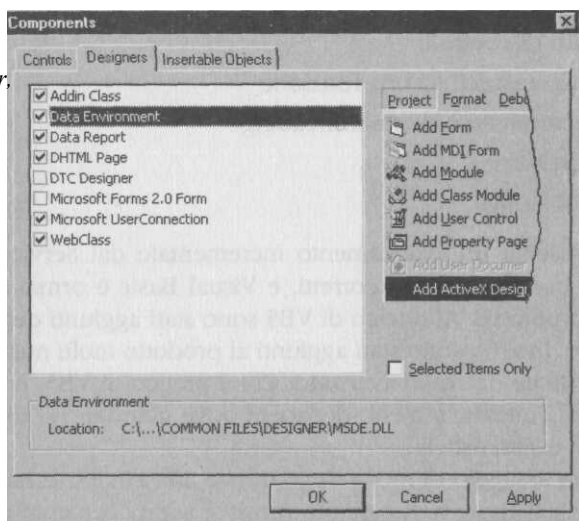
Il Data Environment

Il designer Data Environment (ambiente dei dati) è uno strumento visivo da usarsi in fase di progettazione, che serve a stabilire il comFortamento in fase di esecuzione degli oggetti di dati ActiveX (ActiveX Data Objects, o ADO). Per ulteriori informazioni sugli ADO e sul Data Environment, vedere il Capitolo 32.

In fase di progettazione, usando il Data Environment si può:

- Impostare valori di proprietà per oggetti *Connection,* che controllano la relazione tra una sorgente di dati e l'applicazione.
- Impostare valori di proprietà per oggetti *Command,* che sono basati su procedure memorizzate (stored procedure), tabelle (table), viste (view), o istruzioni SQL.
- Scrivere del codice per rispondere agli eventi ADO.
- Eseguire comandi, creare degli *aggregati* (che sono un insieme di dati provenienti da più righe di una tabella), e stabilire gerarchie (relazioni tra tabelle e sottotabelle), tramite il proprio codice.

Si possono anche legare oggetti Data Environment a controlli o prospetti semplicemente trascinando il Data Environment su un form (che lo lega a un controllo) o sul designer DataReFort (prospetto di dati). Per utilizzare un Data Environment, dapprima ci si assicuri che il designer sia stato abilitato sulla scheda *Designers* della finestra di dialogo *Components,* come mostrato nella Figura 5.1.

Figura 5.1
Prima di poter usare un designer, deve venire abilitato sulla scheda Designers della finestra di dialogo Components del progetto.

Dopo che è stato abilitato, si può aggiungere il designer Data Environment al proprio progetto selezionando *More ActiveX Designers* dal menu *Project.* Quando il designer viene aggiunto al proprio progetto, si apre automaticamente la finestra di dialogo *Properties* relativa al primo oggetto *Connection* del proprio Data Environment, come mostrato in Figura 5.2.

Gli oggetti Connection *del Data Environment servono a impostare la sorgente dei dati.*

Figura 5.2

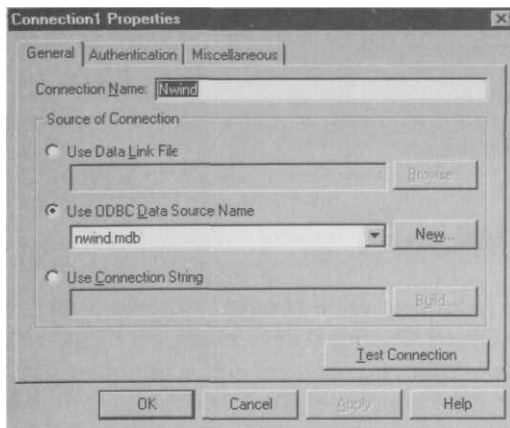
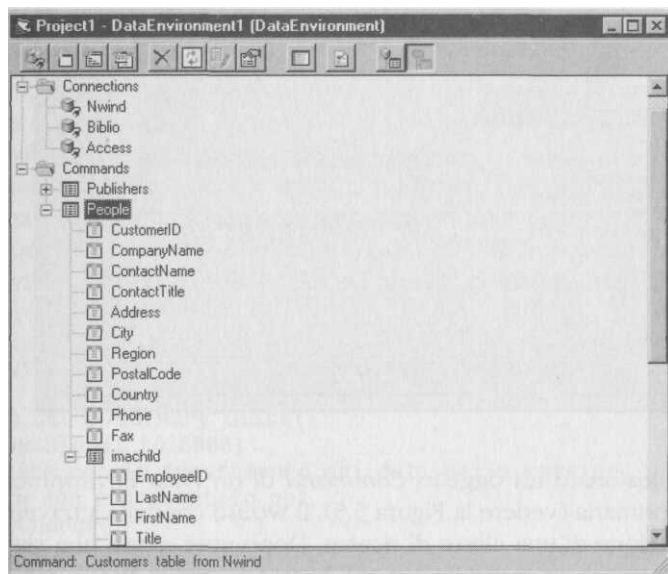
Quando si aggiunge un Data Environment a un progetto, si apre automaticamente una finestra di dialogo Connection Properties per il primo oggetto connessione di tale Data Environment.

Dopo aver impostato il proprio Data Environment con la prima connessione alla sorgente di dati, si possono aggiungere ulteriori oggetti *Connection* e oggetti *Command.* Si può usare la toolbar del Data Environment (mostrata in Figura 5.3) per manipolare le relazioni tra gli oggetti nel designer.

Figura 5.3

Sì può usare la toolbar del Data Environment per manipolare gli oggetti Connection e Command.

È opFortuno notare che, usando il modello a oggetti dell'estendibilità del Data Environment (Data Environment Extensibility Object Model), si può scrivere del codice per manipolare l'oggetto Data Environment in fase di progettazione, cambiandone così la funzionalità. Ciò significa che si può creare un'aggiunta al Data Environment (Data Environment Add-In), che estende il Data Environment. Alcuni esempi potrebbero essere i seguenti:

- Un wizard che creasse un oggetto Data Environment e lo legasse a un forni

- Un'interfaccia alternativa al Data Environment
- Una procedura che restituisse delle informazioni su di un oggetto Data Environment
- Un'estensione di un oggetto *Connection* per accedere a una sorgente di dati in un modo personalizzato

Il Data Object Wizard

Il Data Object Wizard serve a creare classi e controlli d'utente legati a quelle classi. Prima di poter usare il Data Object Wizard, si deve creare un oggetto *Data Environment* contenente oggetti *Command* che recuperino e manipolino dati. Per abilitare il Data Object Wizard, lo si deve caricare nell'Add-In Manager. Il wizard può poi venire avviato dal menu *Add-Ins.* Come mostrato in Figura 5.4, il wizard può venire usato per creare una classe a cui altri oggetti possono legare dati, o un controllo d'utente legato a una classe esistente.

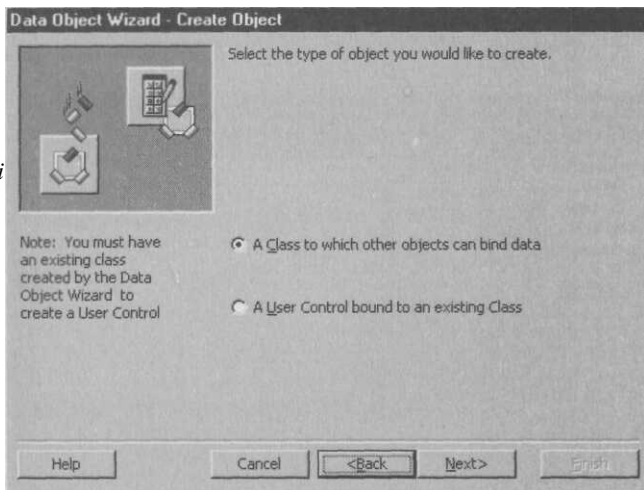
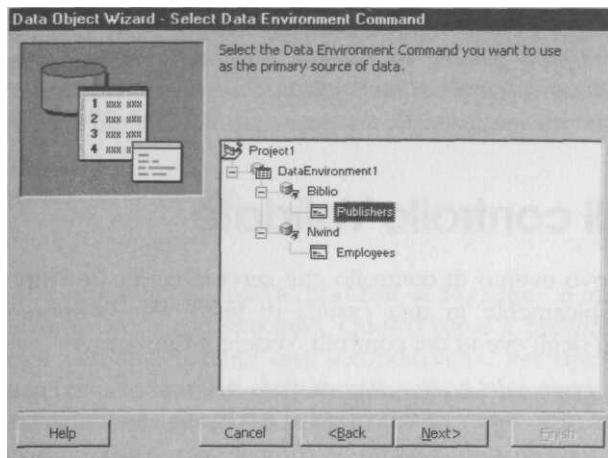
Figura 5.4 *Si può usare il Data Object Wizard per generare una classe per legare oggetti di dati oppure un controllo l'utente legato a una classe esistente.*

Dopo aver selezionato un oggetto *Command* di un Data Environment come sorgente di dati primaria (vedere la Figura 5.5), il wizard conduce attraverso il procedimento di creazione di una classe di ricerca. Dopo aver creato una classe usando il wizard, si può creare un controllo ActiveX basato su quella classe, sempre usando il Data Object Wizard.

Figura 5.5
Prima di creare una classe Data Object Wizard, si deve selezionare un oggetto Cornmand come sorgente di dati primaria.

Controlli persistenti su pagine di Internet Explorer

La persistenza dei controlli ActiveX viene attuata attraverso i due metodi dell'oggetto PropertyBag, ReadProperty e WriteProperty, rispettivamente per leggere e per scrivere i valori di proprietà. Il valore di una proprietà immagazzinata nel PropertyBag può essere esso stesso un oggetto.

Oltre a salvare le proprietà del controllo, si può usare il PropertyBag per far persistere dati binari (come un grafico immagazzinato in un formato personalizzato). Per farlo, si devono immagazzinare le informazioni binarie in una matrice di byte. La matrice di byte deve venire ridimensionata al numero di byte dell'oggetto binario. Per esempio:

```
Private mbytBlob() As Byte ' Dichiara matrice di byte.

Private Sub cmdSaveBinary_Click()
   ReDim mbytBlob(1 to 5000)
     Il codice per lo spostamento dei dati nella matrice
     di byte non è presentato qui
   PropertyChanged
End Sub

Private Sub UserControl_WriteProperties(PropBag As
   PropertyBag)
   PropBag.WriteProperties "myBinary", mbytBlob
End Sub
```

Per ulteriori informazioni sull'utilizzo del PropertyBag per far persistere i controlli ActiveX, vedere il Capitolo 24.

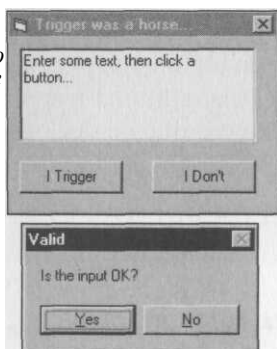
I valori di proprietà per i componenti e controlli ActiveXpossono venire fatti persistere in un PropertyBag *globale in Internet Explorer (versioni 3 e successive), dando la possibilità di salvare dei dati quando un utente naviga fuori da una pagina HTML contenente un controllo d'utente o un documento d'utente.*

L'evento di controllo Validate

Validate è un nuovo evento di controllo che serve a verificare l'input dell'utente con del codice, tipicamente in una casella di testo. Per ulteriori informazioni sull'ordine di scatto degli eventi dei controlli, vedere il Capitolo 3.

Chi avesse provato a Forre del codice di convalida nell'evento LostFocus *di un controllo, comprenderà il bisogno di* Validate. LostFocus *scatta dopo che lo stato attivo dell'applicazione se ne è andato. È difficile usarlo per convalidare l'input dell'utente.*

Una nuova proprietà di controllo, CausesValidation, *funziona con l'evento* Validate. CausesValidation *vale* True *di default. Ma se viene impostata a* False, *per esempio su un pulsante di comando, gli eventi* Validate *non scattano. Una piccola applicazione, salvata sul CD-ROM col nome Valid. Vbp, mostra come si opera con l'evento* Validate *e con la proprietà* CausesValidation. *L'applicazione, mostrata in Figura 5.6, consiste in una casella di testo e in due pulsanti di comando, di nome* itrigger *(io faccio scattare) e* idont *(io no).*

Figura 5.6
Il nuovo evento Validate serve controllare l'input dell'utente nei controlli

L'idea, in questa piccola applicazione, è di Forre del codice nell'evento Validate della casella di testo, e guardare come viene fatto scattare. Come si potrebbe sospettare, siccome la proprietà CausesValidation del pulsante idont è stata impostata a False, fare clic sul pulsante idont non fa mai scattare l'evento. L'altro pulsante, itrigger, ha la sua proprietà CausesValidation impostata a True, perciò viene eseguito il codice Validate della casella di testo. Nel codice, invece di una vera convalida dell'input, viene usata una casella di messaggio:

```
Private Sub Text1_Validate(Cancel As Boolean)
Dim answer As Integer
answer = MsgBox("L'input va bene?", vbYesNo)
```

```
If answer = vbYes Then
   Cancel = False
   itfigger.CausesValidation = False
    'aggiunto per bloccare la seconda chiamata
   Form1 .BackColor = vbRed
Else
   Cancel = True
   Form1.BackColor = vbBlue
End If
End Sub
```

Si noterà che la proprietà CausesValidation di itrigger è impostata a False se l'input viene convalidato con successo. Questo serve a prevenire che il codice di Validate scatti automaticamente una seconda volta. Per ripristinare la proprietà CausesValidation allo stato di default, deve venire aggiunta una riga di codice all'evento GotFocus della casella di testo:

```
Private Sub Text1_GotFocus()
   itrigger.CausesValidation = True
End Sub
```

Per bloccare un utente su un controllo come una casella di testo, si imposta Cancel *a* True *nell'evento* Validate *del controllo:*

```
Private Sub Text1_Validate(Cancel As Boolean)
   Cancel = True
End Sub
```

Aggiunta dinamica di controlli

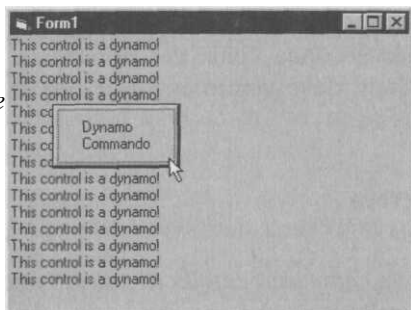
All'insieme di controlli di un forni si può aggiungere dinamicamente un controllo (per ulteriori informazioni sugli insiemi, vedere il Capitolo 14). Per aggiungere un controllo a un form per mezzo del codice, si usi il metodo Add dell'oggetto insieme di controlli:

```
object.Add (ProglD, name, container)
```

Solitamente si può determinare il ProglD di un controllo usando l'Object Browser, specificando la libreria seguita da un punto e dalla classe. Per esempio, il ProglD di un pulsante di comando è "VB.CommandButton". Il parametro successivo, *name, è* un identificatore obbligatorio. L'ultimo parametro, container, specifica il contenitore, per esempio un controllo cornice, in cui inserire il controllo. Se è omesso, per default si intende il form attivo. Si potrebbe usare il codice seguente per aggiungere un pulsante di comando a un form, posizionare il pulsante e stampare del testo sul form (vedere la Figura 5.7):

```
  Dichiara una variabile oggetto come CommandButton.
Private WithEvents cmdObject As CommandButton

Private Sub Form_Load()
   Set cmdObject = Form1.Controls.Add
      ("VB.CommandButton", "cmdOne")
```

```
    cmdObject.Visible = True
    cmdObject.Caption = "Dynamo Commando'
    cmdObject.Height = 750
    cmdObject.Width = 1500
    cmdObject.Top = 800
    cmdObject.Left = 500
End Sub

Private Sub cmdObject_Click()
    Print "This control is a dynamo!"
End Sub
```

Figura 5.7 *Si possono aggiungere dinamicamente controlli a un form, o a un altro contenitore.*

Restituire una matrice da una funzione

Le funzioni e le procedure Property adesso possono restituire matrici. Per restituire una matrice da una funzione, si usa la parola chiave ParamArray (matrice di parametri) nell'elenco di argomenti della funzione. Non si può usare ParamArray in combinazione con ByVal, ByRef, o Optional. La parola chiave ParamArray significa che il parametro specificato dopo il suo uso (l'argomento finale) è una matrice opzionale di elementi varianti. Per esempio:

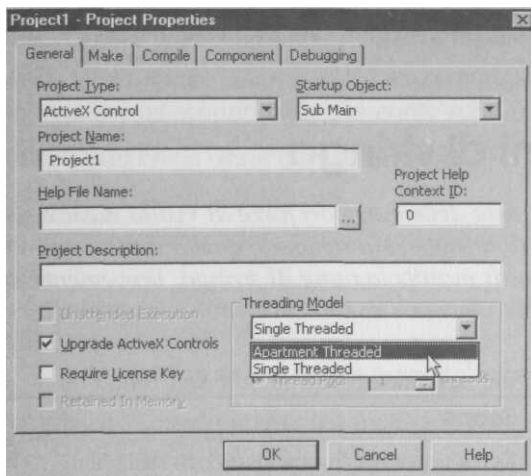
```
Public Function myFunc (ParamArray the_Array())
```

Il modello ad appartamento di multithreading

Diversamente dalle precedenti versioni di Visual Basic, i progetti di Visual Basic 6 possono utilizzare il modello "ad appartamento" di multithreading senza dover sopprimere degli elementi visivi come i forni o i controlli.

Nel modello ad appartamento di multithreading, ogni thread è come un appartamento, nel senso che tutti gli oggetti creati sul thread vivono nell'appartamento, ma non sanno nulla di quello che succede negli altri appartamenti.

La scheda *General* della finestra di dialogo *Project Properties* serve a impostare il modello di multithreading per un progetto di DLL ActiveX, di EXE ActiveX, o di controllo ActiveX, come mostrato nella Figura 5.8.

Figura 5.8 *Si può impostare il modello di multithreading di un'applicazione e usando la scheda General della finestra di dialogo Project Properties.*

Per i progetti di DLL ActiveX e di controlli ActiveX, si può selezionare sia Apartment Threaded *(con multithreading ad appartamento) che* Single Threaded *(a thread singolo). Per i progetti EXE ActiveX, si può o specificare che ogni nuovo oggetto sia creato in un nuovo thread* (Thread per Object), *o limitare il proprio server à un pool fissato di thread. Se la dimensione del pool di thread è posta a uno, il progetto risulta a thread singolo; se la dimensione del pool di thread è maggiore, il progetto risulta con multithreading ad appartamento.*

La funzione CallByName

Si può usare la funzione CallByName per impostare od ottenere un membro di un oggetto basandosi sul nome del membro, passato come argomento di stringa. La forma generalizzata della funzione CallByName è

```
CallByName (object, member_string, calltype Constant,
   optional arguments)
```

dove *member_string* (stringa di membro) è un'espressione di stringa avente come valore il nome di una proprietà o metodo che appartiene all'oggetto specificato nella funzione, e *calltype Constant* (costante di tipo di chiamata) è un membro di VBA.VbCallType, i cui possibili valori sono VbGet, VBLet, VbMethod, e VbSet. Per esempio, il seguente codice sposta una casella di testo alla posizione specificata per mezzo del metodo Move del controllo:

```
Private Sub cmdMove_Click()
   CallByName Text1, "Move", VbMethod, 10, 10
End Sub
```

Volendo cambiare il valore della proprietà MousePointer del controllo per modificare il cursore sulla casella di testo, si potrebbe usare questo codice:

```
Private Sub cmdPoint_Click()
   CallByName Text1, "MousePointer", VbLet, vbHourglass
End Sub
```

Nuove funzioni di stringa

Fin dai suoi primi giorni, una delle maggiori forze di Visual Basic è sempre stata la praticità nel maneggiare le stringhe. Mantenendo quella tradizione, VB6 introduce numerose nuove funzioni di manipolazione di stringa, brevemente descritte nella Tabella 5.1.

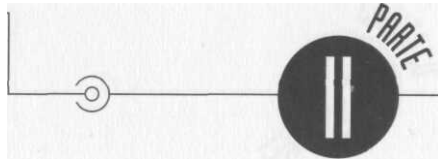
Tabella 5.1 *Funzioni di manipolazione di stringa (nuove in VB6).*

Funzione	**Scopo**
Filter	Cerca un valore in una matrice di stringhe.
FormatCurrency	Rende un'espressione formattata come valuta usando le impostazioni di valuta introdotte nella scheda *Valuta (.Currency)* dello strumento *Impostazioni internazionali (Regional Settings)* nel Pannello di controllo (Control Panel) di "Windows.
FormatDateTime	Rende un'espressione formattata come data o come ora.
FormatNumber	Fornisce varie opzioni per restituire espressioni numeriche formattate.
FormatPercent	Fornisce varie opzioni per restituire espressioni numeriche formattate come percentuali (moltiplicate per 100 e seguite dal carattere %).
InstrRev	Restituisce la prima occorrenza di una stringa entro un'altra stringa, leggendo da destra a sinistra, ossia nell'ordine inverso rispetto al normale.
Join	Congiunge in una sola stringa le stringhe che sono elementi di una matrice monodimensionale. Le sottostringhe della stringa finale possono venire separate da un delimitatore opzionale.
MonthName	Restituisce una stringa con il nome del mese. Per esempio, `MonthName(1 )` rende "January". Il nome del mese può venire reso in forma abbreviata impostando a `True` il secondo parametro della funzione.
Replace	Restituisce una stringa in cui una sottostringa specificata è stata sostituita, un numero specificato di volte, da un'altra stringa.
Round	Restituisce un numero arrotondato al numero specificato di posti decimali.
Split	Divide una stringa in una matrice monodimensionale usando un delimitatore. Il carattere delimitatore di default è uno spazio (" ").
StrReverse	Restituisce una stringa in cui i caratteri sono stati rovesciati.
WeekDayName	Restituisce una stringa per il giorno della settimana specificato. Si può specificare se restituisce il giorno in forma abbreviata, e con quale giorno debba cominciare la settimana. Per esempio, usando i default. `WeekDayName(1 )` restituisce "Monday".

Riepilogo

Questo capitolo ha esplorato alcune delle migliori nuove caratteristiche di VB6. Senza dubbio, ognuno ne scoprirà altre per conto suo!

- Abbiamo spiegato come usare il Data Environment.
- Abbiamo visto il Data Object Wizard.
- Abbiamo trattato come rendere persistenti i dati tramite i controlli usati sul Web.
- Abbiamo dimostrato il nuovo evento di controllo Validate e la proprietà CausesValidation.
- Abbiamo visto come aggiungere dinamicamente controlli a un form.
- Abbiamo spiegato come rendere una matrice da una funzione.
- Abbiamo trattato il modello ad appartamento di multithreading.
- Abbiamo visto come usare la funzione CallByName per invocare un membro di oggetto come letterale di stringa.
- Abbiamo descritto le nuove funzioni di stringa di VB6.

PROGRAMMAZIONE WINDOWS

INTRODUZIONE AI SISTEMI OPERATIVI

FINESTRE DI DIALOGO COMUNI DI WINDOWS

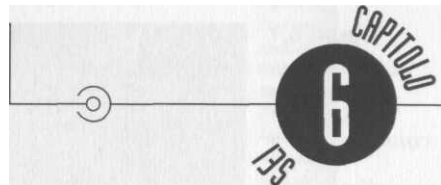
CONTROLLI D'INTERFACCIA UTENTE

USO DEL REGISTRO DI CONFIGURAZIONE

PROGRAMMAZIONE DEL REGISTRO

VISUAL STUDIO API WIN32 E MESSAGGI

VISUAL SOURCESAFE (ENTERPRISE EDITION)

INTRODUZIONE AI SISTEMI OPERATIVI

Le linee guida di Windows

La shell di Windows

I fogli proprietà

I wizard

La finestra dell'applicazione Visual Basic

Utilizzo di ActiveX

Driver dei dispositivi virtuali, macchine virtuali e multithreading

I programmi di installazione

I file di aiuto

Questo capitolo presenta l'interfaccia di Windows, talvolta identificata dal punto di vista del programmatore di Visual Basic con il termine "shell di Windows". La shell di Windows è imFortante per gli sviluppatori in quanto gli utenti finali preferiscono avere a che fare con applicazioni che riprendono l'aspetto e lo stile di questa interfaccia.

Le linee guida di Windows

Dal momento della prima uscita di Windows 95 si sono sviluppate tre diverse tendenze:

- Miglioramento in termini di prestazioni delle successive versioni di Windows 95.
- Progressiva congiunzione tra Windows e Web.
- Implementazione dell'interfaccia amichevole tipica di Windows 95 sulle piattaforme più robuste costituite dalle varie edizioni di NT 4 (poi di Windows 2000).

La congiunzione tra Windows e Web è stata realizzata mediante la funzione Active Desktop di Internet Explorer (versione 4 e successive). Nella Figura 6.1 si può notare che le cartelle dei file vengono rappresentate da collegamenti ipertestuali; alle cartelle si può accedere con un singolo clic (così come nel Web) invece che con un doppio clic (come nella vecchia interfaccia di Windows).

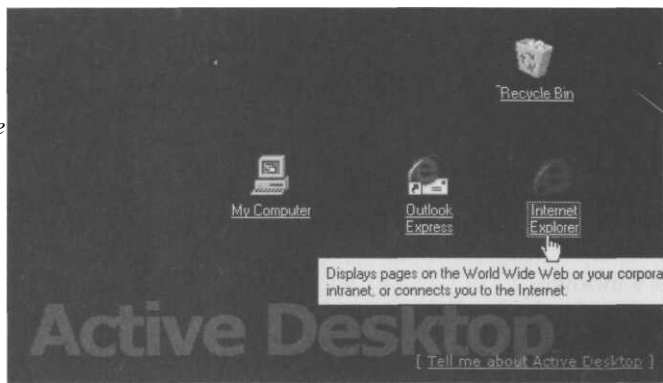
Figura 6.1
L'interfaccia di Windows "in stile Web" consente di fare clic sui collegamenti ipertestuali per accedere applicazioni presenti sul computer.

In sostanza Windows 98 racchiude i miglioramenti successivi di Windows 95 e le funzioni Active Desktop dell'interfaccia Web di Windows. La Figura 6.2 mostra l'aspetto di Active Desktop presente in Explorer (Esplora risorse) di Windows 98, che può essere confrontato con l'Explorer vecchio stile di Windows, visibile in Figura 6.3.

Figura 6.2
Explorer Windows 98 prevede applicazioni accesso ai file in stile Web.

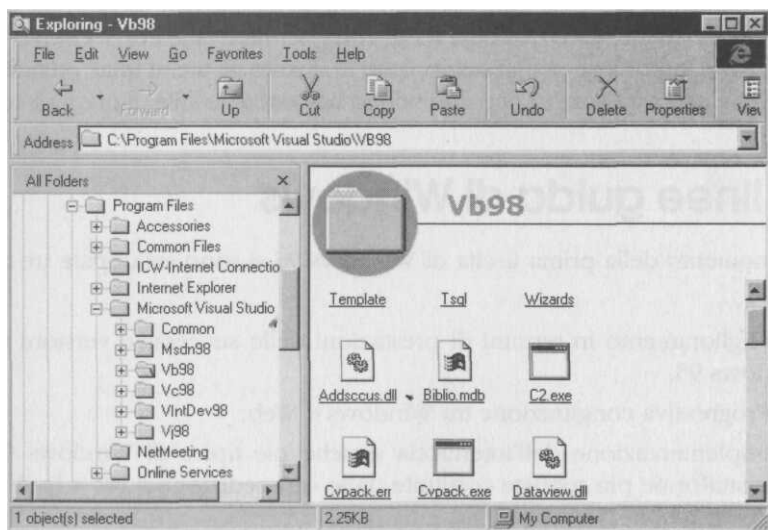
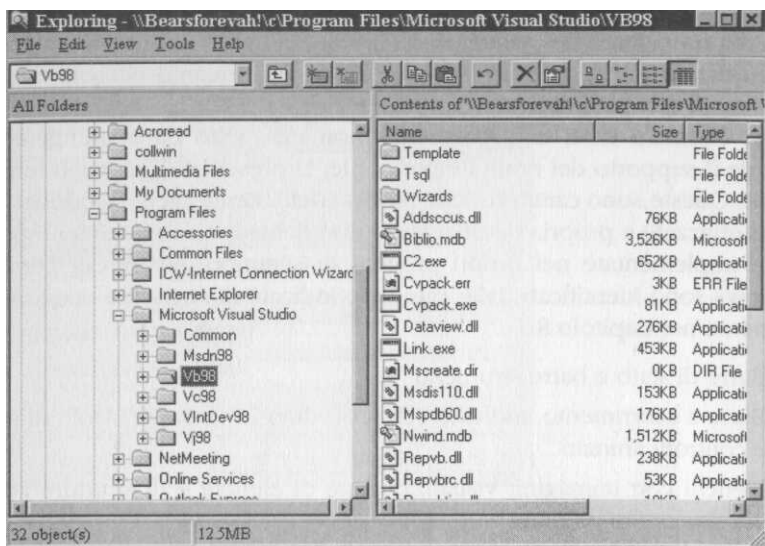



Figura 6.3
Explorer di Windows 95 visualizza file e cartelle con il vecchio stile.

La shell di Windows

Gli sviluppatori hanno a disposizione molti degli elementi che sono stati utilizzati per implementare la funzionalità e l'aspetto generale della shell di Windows. La Professional Edition di Visual Basic comprende la maggior parte dei controlli utilizzati per definire l'interfaccia utente di Windows. In effetti, gli sviluppatori professionisti sono tenuti a creare applicazioni che prevedano aspetto, funzionamento e stile propri di Windows: in parole povere, è un'esigenza sentita profondamente dagli utenti. Gli elementi più imFortanti dell'interfaccia di Windows 95 si possono racchiudere nelle seguenti categorie:

- Controlli personalizzati
- Dialoghi generici
- SupForto dei nomi lunghi di file
- Presenza di scorciatoie
- Anteprima di file
- Menu e fogli delle proprietà attivati con il pulsante destro del mouse
- Possibilità di personalizzare l'interfaccia di Windows

È opportuno notare che i nomi lunghi dei file possono aiutare gli utenti nel riconoscimento preventivo del contenuto di un file, ma gli sviluppatori devono fare attenzione quando intendono prevederne l'impiego in quanto possono nascere diversi problemi di compatibilita.
Tra gli elementi citati in precedenza, non c'è molto da aggiungere per quanto riguarda il supForto dei nomi lunghi di file, la presenza di scorciatoie e l'anteprima dei file. Queste sono caratteristiche proprie dell'interfaccia di Windows che possono essere utilizzate a propria discrezione e non richiedono accorgimenti particolari per essere implementate nei propri progetti di sviluppo. I controlli personalizzati di Windows sono identificati dalle categorie indicate di seguito e vengono trattati diffusamente nel Capitolo 8:

- Barre di stato e barre strumenti
- Barre a scorrimento, indicatori di procedura in corso, controlli di selezione e controlli animati
- Elenchi con immagini, visualizzazione di elenchi e di strutture ad albero, intestazioni di colonna
- Schede e fogli delle proprietà
- Controlli del testo .Rtf

Molti di questi controlli risultano familiari perché si incontrano normalmente nelle applicazioni Windows. Per esempio, Explorer di Windows 95 ne *utilizza* un gran numero; nella schermata di Explorer di Figura 6.3 è possibile vedere l'intestazione di colonne, la visualizzazione a elenco, la barra di stato e i controlli della barra degli strumenti. È possibile integrare nelle applicazioni VB i controlli tipici di Windows e le loro funzioni, come verrà discusso nel Capitolo 8.

I fogli delle proprietà

I fogli delle proprietà sono imFortanti in Windows dato che praticamente sono previsti da tutti gli oggetti. Il foglio delle proprietà di un oggetto, che può essere aperto selezionando la voce *Properties* nel menu di scelta rapida dell'oggetto stesso, è costituito da una serie di finestre di dialogo che contengono informazioni relative all'oggetto.
Per esaminare un semplice esempio di foglio delle proprietà è possibile vedere le informazioni relative a un eseguibile con la finestra di dialogo *Project Properties* di VB, mostrata in Figura 6.4; vi si accede con un clic sul pulsante *Options* nella finestra *Make* oppure scegliendo *Properties* dal menu *Project.* La Figura 6.4 visualizza la finestra *Project Properties* relativa al programma Flow del Capitolo 4.

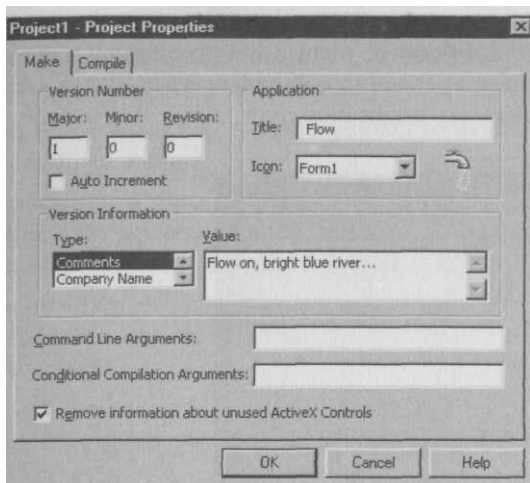
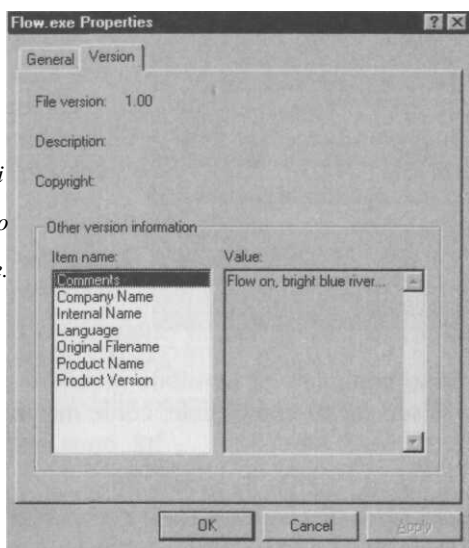
Figura 6.4
Èpossibile incorForare, in un eseguibile compilato, una descrizione e informazioni di versione.

Una volta che il programma è stato compilato, è possibile accedere al corrispondente foglio proprietà mediante il suo menu contestuale, come mostrato in Figura 6.5.

Figura 6.5
Èpossibile accedere alfoglio dette proprietà di un oggetto, per esempio unprogramma eseguibile, selezionando Properties dal menu di scelta rapida dell'oggetto stesso.

Se si seleziona la scheda *Version* nel foglio proprietà relativo a un eseguibile è possibile vedere le informazioni descrittive incluse nel programma, come mostrato in Figura 6.6. È possibile notare per esempio come viene visualizzato il valore inserito in *Comments,* la stringa "Flow on, bright blue river...".

Figura 6.6 ***Informazioni*** *descrittive e relative alla versione diuneseguibile sonodisponibili nella scheda Veriondelfoglio delleproprietà corrispondente.*

Un programma Visual Basic compilato prevede automaticamente un proprio foglio delle proprietà "esterno". Il Capitolo 8 illustra come si possono creare "fogli delle proprietà" in Visual Basic, ai quali gli utenti possono accedere con un clic destro sugli oggetti all'interno del programma stesso.

Wizard

I wizard (procedure guidate) costituiscono una metafora standard dell'interfaccia che viene utilizzata per guidare gli utenti durante lo svolgimento di compiti complessi. In parole povere i wizard sono un genere particolare di fogli proprietà a schede, i quali prevedono un meccanismo che consente all'utente di percorrere un gruppo di finestre di dialogo collegate tra loro.
Esattamente come è possibile mettere a disposizione degli utenti dei programmi un wizard che faciliti lo svolgimento di compiti complessi, Visual Basic 6 dispone di molti wizard che aiutano a sistemare la parte più fastidiosa del lavoro che si incontra durante la creazione dei programmi.

I wizard che vengono forniti con VB6 sono:

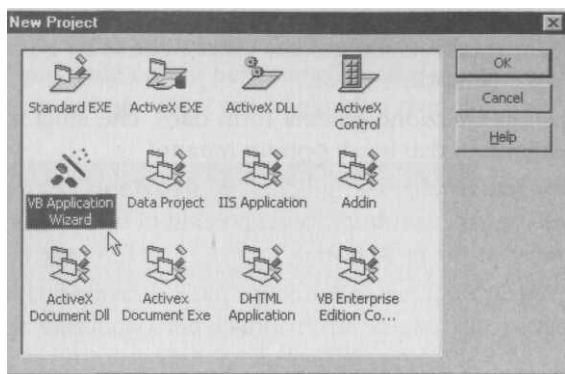
- VB Class Builder Wizard (Creazione guidata classi), che aiuta a implementare le classi e i propri membri.
- VB Data Form Wizard (Creazione guidata form dati), che aiuta a definire moduli basati su sorgenti di dati locali oppure remote.
- VB Application Wizard (Creazione guidata applicazioni VB), che può essere utilizzato per creare la struttura fondamentale di un'applicazione (si veda un esempio nella prossima sezione).
- VB Property Page Wizard (Creazione guidata pagine proprietà), che può essere utilizzato per creare pagine di proprietà personalizzate relative ai controlli ActiveX.
- VB ActiveX Document Wizard (Creazione guidata documenti ActiveX), che traduce form di progetto in documenti ActiveX.
- VB ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX), che può essere utilizzato per definire l'interfaccia dei controlli ActiveX.
- Package and Development Wizard (Creazione guidata pacchetti di installazione), che genera i programmi di installazione delle applicazioni (si veda il Capitolo 35).
- Add-In Designer, che assiste nella creazione di add-in personalizzati (si veda il Capitolo 29).
- Toolbar Wizard (Barra degli strumenti Aggiunte), che si avvia automaticamente quando si aggiunge una barra strumenti a un modulo, per consentire la creazione di barre strumenti personalizzate (si veda il Capitolo 8).

VB6 include anche un gestore di wizard (Creazione operazioni guidate), chiamato a volte il wizard Wizard, che aiuta nella creazione dei propri wizard. Questa operazione viene trattata nel Capitolo 8 e nel Capitolo 30.

/ *wizard di VB sono in effetti add-in in Visual Basic, applicazioni server OLEprogettateper interagire con istanze dell'ambiente Visual Basic; per avere maggiori informazioni sugli add-in, si veda il Capitolo 29. Èpossibile vedere un elenco dei wizard disponibili selezionando* Add-In Manager *nel menu* Add-Ins.

Visual Basic Application Wizard

VB Application Wizard (Creazione guidata applicazioni VB), costituisce un metodo molto efficace per accelerare la definizione di un'applicazione. Per avviare il wizard, selezionare *VB Application Wizard* nella finestra *New Project,* come mostrato in Figura 6.7.

Figura 6.7
Per avviare VB Application Wizard, selezionarlo nella finestra New Project.

VB Application Wizard viene mostrato nella finestra New Project *in quanto è presente un file con estensione . Vbz (VB Application Wizard. Vbz) nella directory dei modelli di progetto. VB Application Wizard. Vbz è un file di testo che contiene il nome della classe del programma server OLE che definisce Application Wizard, classe che viene a volte chiamata programmatic ID, oppure ProgID.*

In VB6 sono state ampiamente modificate le caratteristiche e le funzionalità di Application Wizard. Per chi è alle prime armi, le impostazioni di Application Wizard possono essere prelevate e memorizzate in un file di profilo del wizard, con l'estensione .Rwp. La Figura 6.8 mostra il primo pannello di Application Wizard, che può essere utilizzato per selezionare un profilo. Si può notare che la prima volta che si utilizza questo wizard non ci sono profili disponibili; i profili non esistono fino a quando non vengono creati mediante la funzione Save As Profile *di Application Wizard.*

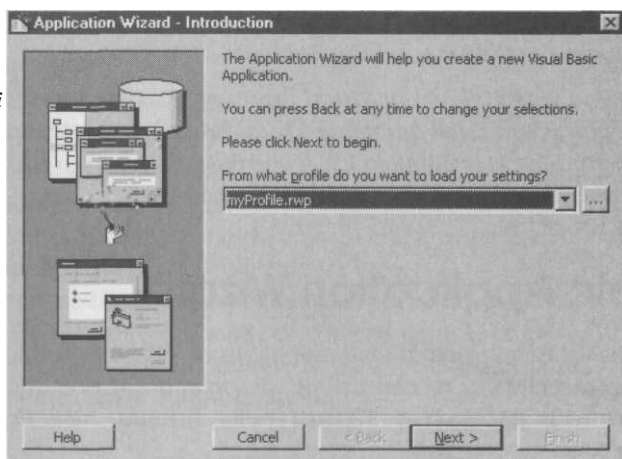
Figura 6.8
È possibile memorizzare le impostazioni Application Wizard in un profilo.

Il passo successivo consiste nell'utilizzare Application Wizard per selezionare il tipo di interfaccia, come mostrato in Figura 6.9. Le opzioni sono l'interfaccia per documenti multipli MDI (Multiple Document Interface), quella per documento singolo SDI (Single Document Interface) e la modalità Explorer Style. Per avere maggiori informazioni sull'interfaccia MDI si veda il Capitolo 18.

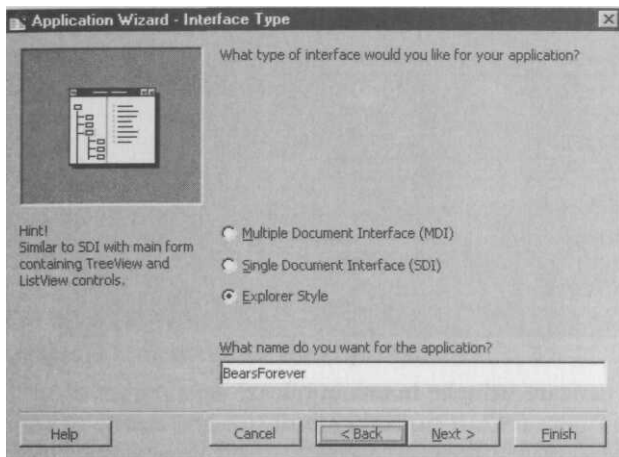
Figura 6.9
Se si seleziona il tipo di interfaccia Explorer Style, Application Wizard inserisce per conto proprio i controlli appropriati in un modulo.

Se si seleziona il tipo di interfaccia Explorer Style, vengono aggiunti automaticamente i necessari controlli e menu al modulo dell'applicazione; questo pannello viene utilizzato anche per definire il nome dell'applicazione.
Il pannello successivo di Application Wizard facilita la personalizzazione di menu e sottomenu, come mostrato in Figura 6.10. Questi menu possono essere modificati successivamente utilizzando il menu editor (si veda il Capitolo 18).

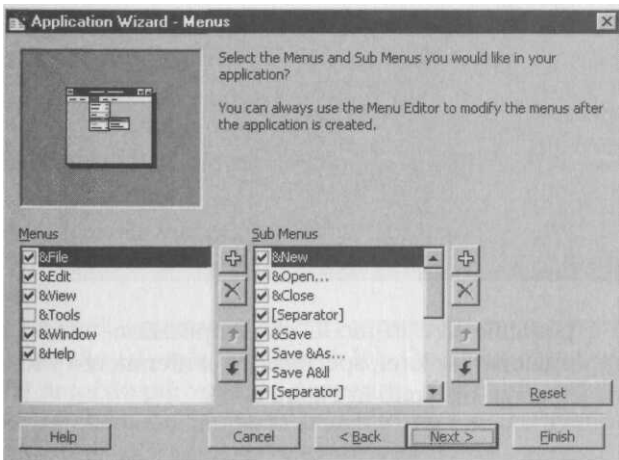
Figura 6.10
Application Wizard aiuta a costruire menu personalizzati.

Il passo successivo riguarda la Toolbar, come mostrato in Figura 6.11.

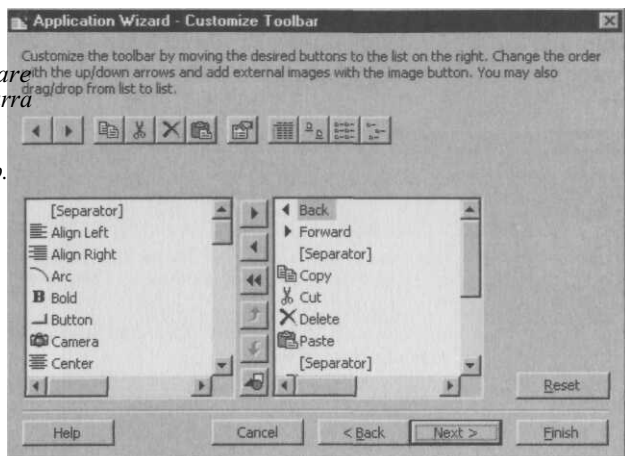
Figura 6.11
È possibile personalizzare la propria barra strumenti utilizzando drag and drop.

È possibile anche caricare stringhe in una applicazione a partire da un file di risorse, come mostrato in Figura 6.12. Il caricamento di stringhe da file di risorse viene utilizzato per creare facilmente versioni in più lingue di un'applicazione. Informazioni su questo argomento si possono trovare nel Capitolo 18.

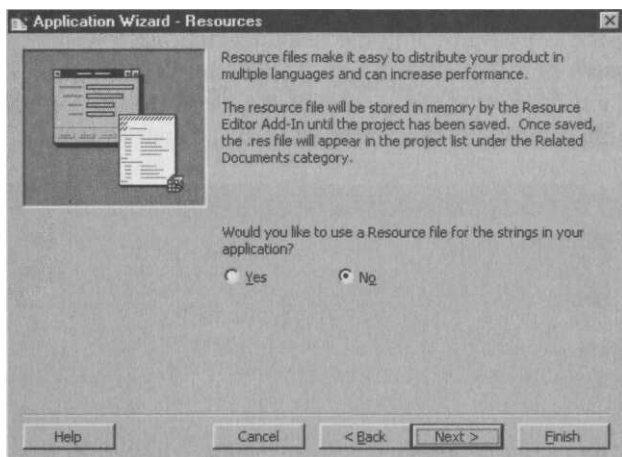
Figura 6.12
Application Wizard consente di caricare stringhe da un file di risorse.

Se lo si desidera, è possibile fare in modo che Application Wizard aggiunga un browser Web all'applicazione Explorer, specificando il riferimento URL di avvio predefinito per il browser, come mostrato in Figura 6.13.

Figura 6.13

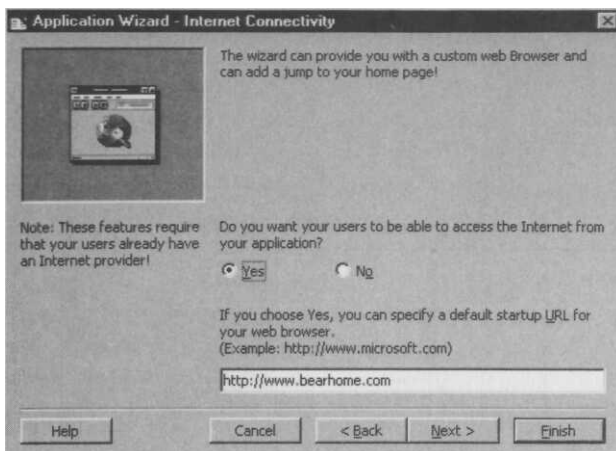
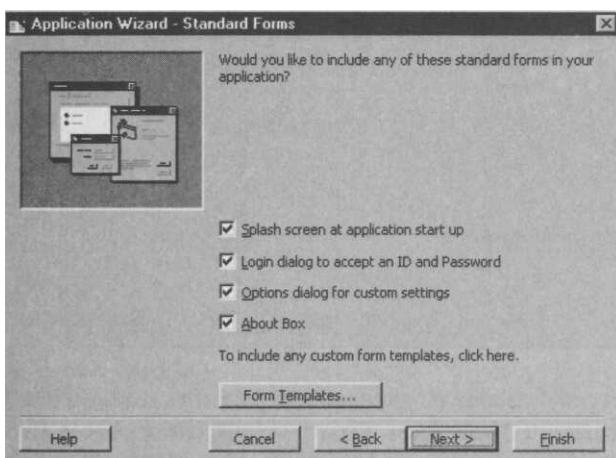
E possibile inserire un brouser Web e specificare l'URL di avvio.

È possibile fare in modo che Application Wizard inserisca moduli aggiuntivi in una applicazione, incluse schermate di tipo Splash, Login e About (si veda la Figura 6.14).

Figura 6.14

È possibile fare in modo che Application Wizard inserisca differenti schermate netta vostra applicazione.

Il pannello successivo del Wizard consente di aggiungere moduli creati con il Data Form Wizard (si veda la Figura 6.15); *è* possibile così creare moduli di accesso dati di solo codice.

La schermata finale di Application Wizard consente di memorizzare le impostazioni correnti in un profilo (si veda la Figura 6.16). Questo profilo può essere utilizzato per replicare il progetto più volte, se necessario. È anche possibile utilizzare questa schermata per ottenere un rapForto generato con suggerimenti specifici in relazione a sviluppi futuri dell'applicazione.

Figura 6.15
Data Form Wizard è stato integrato con Application Wizard.

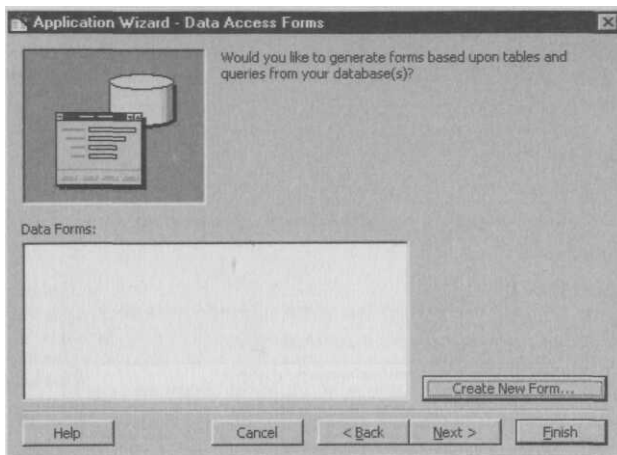



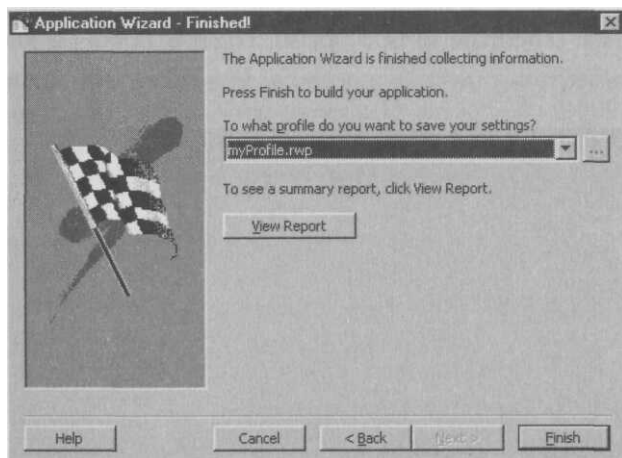
Figura 6.16
È possibile utilizzare l'ultima schermata di Application Wizard per memorizzare le impostazioni del progetto in un profilo e per generare un rapporto.

Da questo punto in poi il volante è nelle mani del Wizard; sulla base di quanto definito nelle diverse schermate di Application Wizard, l'applicazione server ActiveX agisce in modo trasparente e genera gli appropriati moduli e controlli. Può essere necessario un po' di tempo, durante il quale si può osservare quello che sta succedendo. Mi piace pensare a questa fase di lavoro come a quando si osserva un'automobile passare all'interno di un lavaggio automatico, dove non è necessario alcun intervento manuale. Alla fine si ottiene una ossatura abbastanza completa della propria applicazione, dove molto dell'interfaccia utente è già al posto giusto. Se si specifica l'interfaccia Explorer, con funzionalità Internet, il form principale dell'applicazione risultante diventa molto simile a quello mostrato in Figura 6.17 all'interno del corrispondente ambiente di progetto VB6.

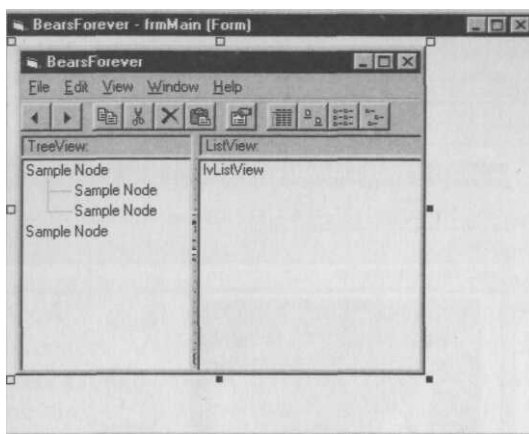
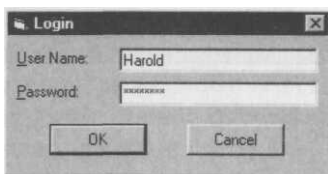
Figura 6.17 *Application Wizard è in grado di generare l'ossatura di un'appHcazione, predisposta con la maggior parte degli elementi dell'interfaccia utente "pronti all'uso".*

Quando si esegue l'applicazione, si comincia con la finestra *Login,* se così è stato specificato nel Wizard, come mostrato in Figura 6.18.

Figura 6.18 *Application Wizard è in grado di creare automaticamente unafinestra Login.*

È facile comprendere che il Wizard non può aggiungere tutta la logica pertinente alla codifica di base; vengono pertanto inseriti commenti nella forma di "cose da fare" (To Do), che stanno a indicare la necessità di inserire altro codice logico. Per esempio, la finestra Login controlla che la password sia vuota; viene quindi richiesta esplicitamente una ulteriore implementazione:

```
Private Sub cmdOK_Click()
    'ToDo: create test for correct password
    'check for correct password
    If txtPassword.Text = "" Then
        OK = True
        Me.Hide
    Else
        MsgBox "Invalid Password, try again!", , "Login"
        txtPassword.SetFocus
        txtPassword.SelStart=0
        txtPassword.SelLength = Len(txtPassword.Text)
    End If
End Sub
```

Se è stato detto al Wizard di aggiungere un browser, quando si esegue l'applicazione si può accedere a questo dal menu *View.* Come mostrato in Figura 6.19, il browser si apre con l'URL di avvio che è stato specificato nel Wizard.

Figura 6.19
Se in Application Wizard si sceglie di aggiungere un browser, questo viene aperto con l'URL che è stato specificato.

ActiveX e Windows

L'esperto di OLE Kraig Brockschmidt afferma che OLE (oppure ActiveX, come si usa dire in Windows) costituisce un terzo del sistema operativo; questa stima può anche essere inferiore al vero. Come minimo, si sta parlando del terzo più significativo.

Nella Parte V di questo libro è mia intenzione fornire tecniche, scorciatoie, segreti e suggerimenti necessari per utilizzare VB con successo al fine di creare programmi pienamente compatibili OLE, tali da sbalordire i vostri utenti. È quindi ovvio che si parlerà di ActiveX diffusamente in quella parte del libro, ma con un obiettivo piuttosto particolare. Nel frattempo conviene soffermarsi su quello che si intende per tecnologia OLE nel suo complesso.

"ActiveX" è il nome con il quale oggi è conosciuta la tecnologia che una volta era chiamata "OLE". ActiveX, oppure OLE, è costituita in effetti da quattro parti distinte:

- Un modello di *documento composto,* che consente la memorizzazione strutturata interpiattaforma per tutti i tipi di informazioni che vengono registrate in un file.
- L'automazione OLE, che consente ad applicazioni compatibili OLE di sviluppare metodi e insiemi di comandi che funzionano all'interno e tra le diverse applicazioni.
- La funzionalità drag and drop OLE, che consente di trascinare oggetti da un'applicazione a un'altra.

I servizi generici OLE, tra cui le interfacce per il trasferimento di dati, le interfacce per la gestione della memoria e per le registrazioni; queste sono solo alcune delle voci che fanno parte di un corposo elenco di servizi.

È chiaro che Windows è stato costruito in gran parte attorno a elementi che fanno un uso massiccio di questi quattro aspetti di ActiveX. L'integrazione di ActiveX in Windows da all'utente "la sensazione che quasi tutto possa essere considerato un oggetto", come afferma Tony Williams, uno dei capi progettisti di Microsoft. È altrettanto evidente che se si desiderano l'aspetto e il feeling propri di Windows, insieme alla filosofia che vi sta alle spalle, si deve implementare ActiveX.
I controlli ActiveX creati in VB possono essere utilizzati da qualunque altro ambiente contenitore ActiveX. Il contenitore del controllo mette a disposizione, ovvero espone, proprietà, eventi e metodi ActiveX. Per avere maggiori informazioni sulla creazione di controlli ActiveX in VB, si veda la Parte VI.

Altre informazioni sui sistemi operativi Windows

Esistono altre caratteristiche interne del sistema operativo Windows che risultano imFortanti per quei programmatori il cui lavoro deve interagire con l'ambiente. Uno degli elementi più imFortanti di Windows è costituito dal Registro di configurazione centrale, trattato dettagliatamente nel Capitolo 9 e nel Capitolo 10. In questa sezione vengono introdotti altri nuovi aspetti imFortanti di Windows: i driver dei dispositivi virtuali, le macchine virtuali e il multithreading.

Driver dei dispositivi virtuali

Windows è costruito attorno a un sistema di driver di dispositivi virtuali VxD (Virtual Device Drivers) i quali gestiscono l'interazione tra programma e sistema operativo con le diverse categorie di dispositivi. Un VxD è un driver in modalità protetta a 32 bit che gestisce una risorsa di sistema, in un modo che consente a più di un programma alla volta l'utilizzo della risorsa. La lettera *x* identifica il tipo di driver; per esempio, VPD è il driver di dispositivo virtuale che gestisce un dispositivo di stampa.
Uno dei vantaggi di questo modo di operare consiste nel fatto che i fornitori di dispositivi (per esempio, i produttori di una stampante) devono fornire un numero limitato di informazioni aggiuntive (indicate con il termine "mini driver") necessarie per un dispositivo particolare. In sostanza un mini driver si connette al VxD.
Questo contrasta nettamente con la situazione che si aveva con Windows 3.x. In generale i driver di Windows 3.x erano complessi e riguardavano un singolo dispositivo, il che Fortava a una duplicazione del codice di gestione di un dispositivo sui sistemi finali, a problemi e costi per i fornitori dei dispositivi e a maggiori probabilità di guasti nei driver dei dispositivi.

Macchine virtuali

Una macchina virtuale VM (Virtual Machine) definisce un ambiente in memoria che viene considerato da un'applicazione in esecuzione come un computer a se stante. Virtual Machine Manager di Windows fornisce una VM per ogni applicazione in esecuzione, corredata delle risorse di sistema necessarie per quella particolare applicazione.
Uno dei vantaggi legati all'esecuzione di più applicazioni, ciascuna con la propria VM, consiste nel fatto che quando un'applicazione si blocca (per esempio, a causa di un errore di protezione), nella maggior parte dei casi si interrompe solo la VM relativa all'applicazione. Questo pone fine al tremendo messaggio di Windows 3.1 "General Protection Fault" che compariva in una finestra con il pulsante *OK,* quando di OK non c'era proprio nulla.

Multithreading

Il multithreading è un meccanismo che consente di eseguire diverse applicazioni simultaneamente. In Windows 3.1 le applicazioni venivano eseguite insieme utilizzando un sistema chiamato *multitasking cooperativo;* questo sistema richiede al sistema operativo di controllare periodicamente la coda di messaggi e di cedere il controllo del sistema alle diverse applicazioni in esecuzione. Si potevano facilmente scrivere applicazioni che risultavano "ingorde" per il semplice fatto che non effettuavano spesso il controllo della coda di messaggi. Per ragioni di compatibilita all'indietro, Windows 95 lavora in multitask cooperativo nel caso di applicazioni a 16 bit.
D'altro canto le applicazioni a 32 bit vengono eseguite in Windows 95 in modo *multitasking preemptive.* Questo significa che il sistema operativo valuta le necessità di ciascuna applicazione in esecuzione e alloca le risorse in modo appropriato. Dato che le applicazioni a 32 bit non devono prevedere di lasciare il controllo ad altre attività in esecuzione, queste possono trarre vantaggio dalle funzionalità multithreading presenti nelle versioni a 32 bit di Windows.
Ogni applicazione multithreading eseguita in modo concorrenziale viene chiamata un *processo.* Ogni processo contiene uno o più *thread,* unità di codice che definiscono una Forzione di tempo *(time slice,* una singola allocazione del tempo di esecuzione disponibile) del sistema operativo e che viene eseguita in modo concorrenziale rispetto agli altri thread. Un'applicazione a 32 bit può sempre iniziare più di un thread per processo (in effetti, ne può attivare fino a 255), aumentando così la velocità apparente dell'applicazione e consentendo l'effettiva elaborazione di attività in background.

Programmi di installazione

Package and Deployment Wizard, fornito con VB6, è in grado di creare automaticamente nella maggior parte dei casi l'appropriata routine di installazione; questo è il wizard che in precedenza era chiamato Setup.

I requisiti per i programmi di installazione a 32 bit sono:

- SupForto dei nomi lunghi di file.
- Registrazione dell'applicazione e dell'estensione.
- Utilizzo del Registro di configurazione al posto dei file di inizializzazione per conservare informazioni tra una sessione e la successiva.
- Creazione di collegamenti nel menu *Start* (al posto di gruppi e icone di Program Manager).
- Creazione di un'utility per la rimozione automatica dell'applicazione.
- Creazione di un file log che stabilisca con precisione quello che è stato fatto dalla routine di installazione.

Oltre a questo, gli utenti di Windows a 32 bit sembrano gradire molto la funzione di autorun delle installazioni da CD-ROM (autorun avvia automaticamente un'installazione basata su CD-ROM nel momento in cui il disco viene inserito nell'apposito lettore).

Èpossibile creare un'installazione di tipo autorun aggiungendo un file di testo chiamato Autorun.Inf nella directory principale del CD-ROM; si deve definire la riga OPEN di questo file con il nome delprogramma di installazione che deve essere eseguito automaticamente. Per esempio:

```
[AutoRun]
OPEN=setup.exe
```

Package and Deployment Wizard può essere utilizzato anche per lo sviluppo Web delle applicazioni e per i controlli ActiveX. Per avere maggiori informazioni si veda il Capitolo 27; i programmi di installazione vengono trattati nel Capitolo 35.

File di guida

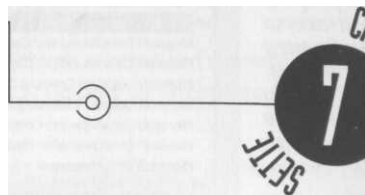
Come ho scritto nella precedente edizione di questo libro, "nel mondo dello sviluppo del software è evidente che niente rimane uguale a se stesso. I file di aiuto sono in continua mutazione; non bisogna essere degli esperti per rendersi conto che le pagine HTML del Web condividono imFortanti caratteristiche con i file di aiuto: entrambi i formati hanno a che fare con testo strutturato ed entrambi presentano collegamenti di tipo ipertestuale. Da questa considerazione si può dedurre che i file di aiuto stanno subendo una graduale (a volte non così graduale) modifica del formato. Non c'è dubbio che tra qualche anno la maggior parte dei sistemi di aiuto verrà scritta con il linguaggio HTML".

Bene, mi sembra di avere precorso i tempi con qualcosa che si sta puntualmente verificando! Niente vi può togliere l'idea di creare file di guida vecchio stile e di collegare a questi le vostre applicazioni; Visual Studio 6 e altri distributori di terze parti mettono ancora a disposizione gli strumenti necessari. Tuttavia, le applicazioni più attuali, incluso Visual Basic 6, fanno uso di Help in formato HTML; questo argomento è trattato dettagliatamente nel Capitolo 34.

Riepilogo

Questo capitolo ha introdotto la shell di Windows.

- Ho mostrato come incorForare informazioni sulla versione dei file in un eseguibile Visual Basic, in modo che queste compaiano nel foglio proprietà del programma compilato.
- Vi è stata fornita una presentazione dei wizard disponibili in VB6.
- Ho spiegato come utilizzare il potenziato ed espanso Application Wizard di VB6.
- È stata fornita un'introduzione ad ActiveX.
- Avete appreso molte caratteristiche del sistema operativo Windows 95/98, inclusi i VxD, le macchine virtuali e il multithreading.

FINESTRE DI DIALOGO COMUNI DI WINDOWS

- I sei tipi di finestre di dialogo comuni supFortate dal controllo di Windows
- Costanti e flag del controllo delle finestre di dialogo comuni
- Definizione di proprietà e flag delle finestre di dialogo comuni senza utilizzare codice
- Controllo delle impostazioni sul tipo di file mediante la proprietà .Filter
- Inserimento di flag e proprietà nella codifica
- Spiegazione dell'oggetto VBA FileSystem

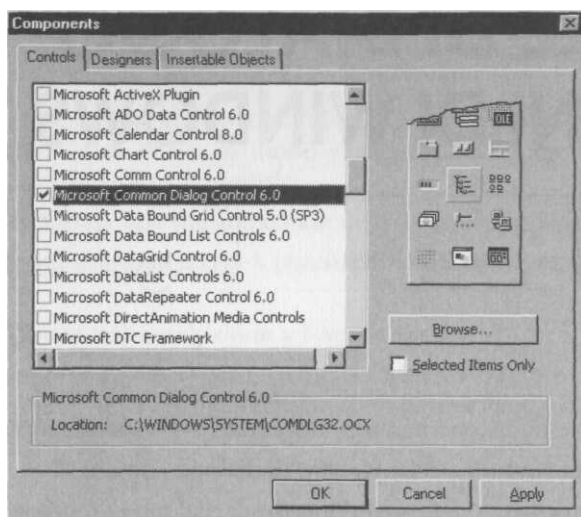
In questo capitolo vedremo come utilizzare la versione 6 del controllo ActiveX Common Dialog; questo controllo è elencato nella finestra *Components* come Microsoft Common Dialog Control 6.0. Di solito viene genericamente indicate come "controllo dei dialoghi comuni".

Finalità del controllo dei dialoghi comuni

È ovvio che il primo passaggio nell'utilizzo del controllo dei dialoghi comuni consiste nell'aggiungerlo al vostro progetto. Per fare questo si utilizza la finestra *Components* per aggiungere Microsoft Common Dialog Control 6.0 alla Toolbox, se non è già presente, come mostrato in Figura 7.1.

La finalità del controllo dei dialoghi comuni consiste nel facilitare agli sviluppator l'inserimento nelle loro applicazioni di finestre di dialogo che presentino aspetto facilità d'uso efunzionalità tipiche di Windows.

Una volta inserito il controllo dei dialoghi comuni nella Toolbox, fare doppio clic per aggiungere il controllo nel form corrente attivo nel progetto. Il controllo dei dialoghi comuni funziona come un involucro *(wrapper),* un intermediario tra Visual Basic e le funzioni di libreria presenti nella libreria a collegamento dinamice Comdlg32.Dll. Questo file deve essere posizionato nella directory Windows\Systems in Windows 98/95 e nella directory WinNT\System32 in Windows NT. Il nome del file relativo al controllo è Comdlg32.Ocx.

Figura 7.1 *È necessario aggiungere il controllo dei dialoghi comuni alla propria Toolbox attraverso la finestra Components, prima di poterlo utilizzare nei progetti.*

L'idea è di facilitare per i programmatori Visual Basic (e altri) l'accesso a queste funzioni utilizzando proprietà di controllo definite "visivamente" durante la progettazione in Property Viewer, oppure mediante una piccola aggiunta in fase di codifica. Uno dei vantaggi legati al fatto di avere dialoghi comuni facilmente disponibili riguarda la standardizzazione; quando l'utente esegue un compito di per sè familiare, come il salvataggio di un file su disco, gli viene chiesto di aprire una finestra di dialogo anch'essa familiare.

Probabilmente è ovvio, ma conviene ribadirlo: le finestre di dialogo non fanno nulla di più che Fortare in primo piano una visualizzazione che permette all'utente di compiere delle scelte. Se l'utente decide di salvare un particolare file, sceglie un determinato font e così via, si deve comunque "attivare" questa decisione; in altre parole occorre aggiungere il codice che esegua il compito richiesto.

Nell'insieme, Comdlg32.Ocx funziona abbastanza bene. Esistono sei dialoghi comuni che possono essere visualizzati in applicazioni Windows mediante il controllo Comdlg32.Ocx (si veda la Tabella 7.1) e il metodo appropriato.

Tabella 7.1 *Dialoghi comuni.*

Dialogo	Metodo che lo visualizza	Equivalenza nella proprietà Action
Color	.ShowColor	3
Font	.ShowFont	4
Apre WinHelp in corrispondenza del file di aiuto *e* dell'argomento specificati	.ShowHelp	6
File Open	.ShowOpen	1
Print	.ShowPrinter	5
File Save As	.ShowSave	2

Corndlg32.Ocx non è un controllo per le finestre, ovvero non è in grado di disegnare per conto proprio nulla sullo schermo, a parte l'icona Toolbox che sistema nel forni, né durante la progettazione e nemmeno in fase di runtime. (Un esempio di semplice controllo per le finestre è definito dal pulsante di comando. L'aspetto in fase di progettazione di un pulsante di comando cambia in funzione delle scelte del programmatore; in runtime il suo aspetto può cambiare in base al progetto, alla programmazione e alle scelte dell'utente.)
Dovrebbe essere noto che non si ha alcuna possibilità di controllo (scusate il gioco di parole) della posizione nella quale Comdlg32.Ocx decide di sistemare il dialogo che l'utente ha aperto.

È stata lasciata la proprietà Action *per garantire la compatibilita all'indietro con Visual Basic Versione 3 (e precedenti). Non c'è in effetti alcuna ragione per utilizzarla ora nella codifica che dovete ancora scrivere. Nonostante questo, se siete curiosi di vedere come lavora, quando definite la proprietà di azione di dialogo comunepotete invocare il dialogo opFortuno come segue:*

```
CommonDialogl.Action = 3
```

che è equivalente a

```
CommonDialogl.ShowColor
```

Costanti e flag del controllo

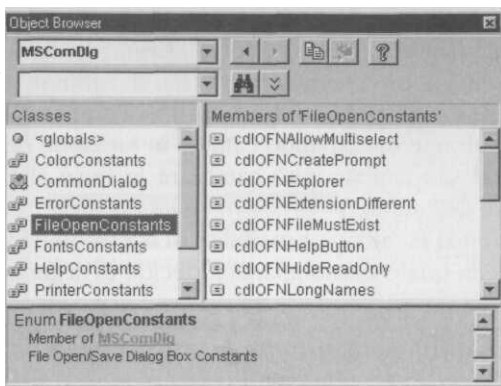
Sia che definiate il controllo dei dialoghi comuni utilizzando la finestra *Properties* oppure in codifica (entrambe le modalità sono descritte più avanti in questo capitolo), alcuni dei più imFortanti aspetti del dialogo che si vuole aprire sono determinati dal valore della corrispondente proprietà .Flags:

```
CommonDialogl.Flags
```

Il valore della proprietà . Flags è definito da interi lunghi che possono essere inseriti uno dopo l'altro per formare una combinazione di attributi (lavora in modo simile alla funzione MsgBox).
I valori possono essere espressi come interi oppure, come si usa nella pratica moderna, per mezzo delle costanti equivalenti. Questa è la procedura consigliata in quanto rende più evidente quello che ci si aspetta dal codice.

Si può utilizzare Object Browser come strumento per determinare le costanti disponibili per i dialoghi comuni (e il loro valore numerico), come mostrato nella Figure 7.2.*

Figura 7.2

È possibile utilizzare Object Browser per definire le costanti dei dialoghi comuni.

Per esempio,

```
CommonDialogl.Flags = cdlOFNOverWritePrompt +
   cdlOFNHelpButton + cdlOFNPathMustExist
CommonDialogl.ShowSave
```

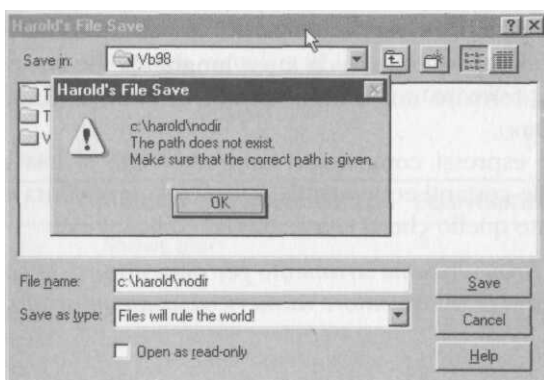
fa aprire una finestra *File Save As* con un pulsante di Help. Questo dialogo genera un messaggio di avvertimento prima di consentire all'utente di selezionare un file esistente, e non permette all'utente di inserire un percorso che non esiste (si veda la Figura 7.3).

È possibile naturalmente utilizzare l'equivalente numerico delle costanti dei flag per ottenere lo stesso risultato:

```
CommonDialogl.Flags = 2066 '2066 = 2 + 16 + 2048
```

Penso ad ogni modo che sarete d'accordo con me che il significato della prima istruzione risulta molto più chiaro.

Figura 7.3

Se è stato definito il flag PathMustExist, gli utenti vedono comparire questo messaggio "path does not exist" (il percorso non esiste) nel caso in cui si tenti di registrare un file utilizzando un percorso inesistente.

Da dove vengono i valori delle costanti relative alla proprietà .Flags? Una volta incluso in un progetto il controllo dei dialoghi comuni, si possono automaticamente utilizzare le espressioni delle costanti nel codice senza preoccuparsi dei loro valori.

È anche possibile utilizzare per il controllo la finestra *Property Pages* (oppure la finestra *Properties)* in fase di progettazione per definire molti valori di proprietà del controllo. Tuttavia, per definire il valore della proprietà Flags nella finestra *Properties* in fase di progettazione occorre sapere il valore numerico di ciascuna costante (si può utilizzare la notazione decimale oppure quella esadecimale). Sono disponibili tre fonti per conoscere valori e significato delle costanti dei flag:

- L'aiuto VB online alla voce *CommonDialog Control Constants*
- L'aiuto VB online alla voce *Flags Property* e per ciascun differente tipo di dialogo
- Object Browser, in corrispondenza di MSComDlg (Microsoft Common Dialog Control), elenca le costanti con i valori corretti interi decimali (ed esadecimali)

Per disForre della finestra Font *che visualizza ifont disponibili, si deve prima definire la proprietà* .Flags *dei dialoghi comuni in modo che includa un valore costante che dica di mostrare ifont. Per esempio,*

```
CommonDialogl.Flags = cdlCFBoth
```

dice al dialogo comune di visualizzare sia i font di schermo sia quelli della stampante. Se prima non si invia al controllo un flag di visualizzazione dei foni, si può spendere un sacco di tempo a chiedersi perché si continua a ricevere un messaggio di errore cdlNoFonts, *quando si sa perfettamente che ci sono molti foni installati sul proprio sistema (si veda la Figura 7.4)!*

Figura 7.4
Nessun font? Proprio nessun font? Ehi, non mi sembra proprio...

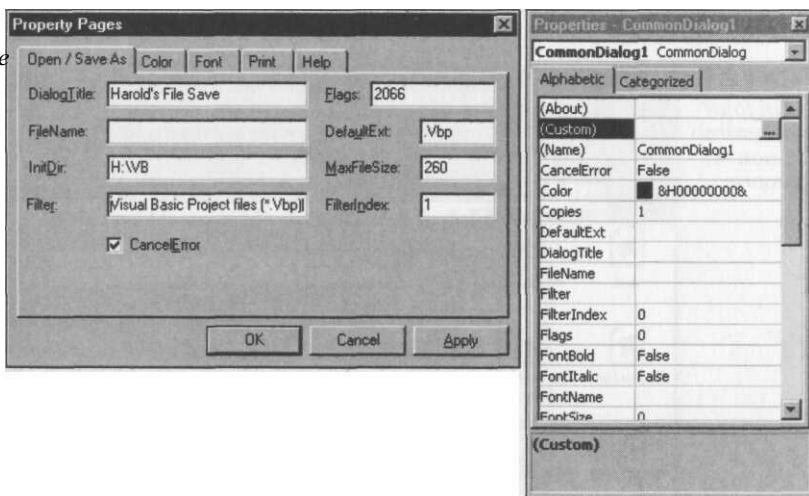
È possibile utilizzare il flag cdlOFNExplorer *(&H80000) per aprire una finestra di dialogo* Open File *in stile Explorer; i dialoghi comuni che utilizzano questo flag non funzionano con Windows NT. Se si cerca di eseguire il metodo* ShowOpen *per il controllo di un dialogo comune definito con questo valore di flag, si ottiene un messaggio di errore che indica come "l'oggetto non supForta questo metodo o proprietà ".*

Guarda mamma, niente codice

È facile utilizzare la finestra *Properties* per definire proprietà e flag dei dialoghi comuni senza utilizzare una riga di codice; si devono solo stabilire le proprietà, e lo si può fare nella finestra *Properties* oppure mediante il dialogo *Property Pages* personalizzato.

Si accede al dialogo Property *Pages personalizzato per un controllo facendo clic sul pulsante nel campo valore relativo alla proprietà* Custom *nella finestra* Properties *(si veda la Figura 7.5) oppure scegliendo la voce* Properties *del menu di scelta rapida del controllo selezionato.*

Si può notare che i valori inseriti in un modo vengono automaticamente resi disponibili anche nell'altro. Per esempio, se si modifica la proprietà .DialogTitle in "Harold's File Save", quando si apre il dialogo personalizzato si può osservare che la casella di input *DialogTitle* contiene anch'essa il medesimo valore.

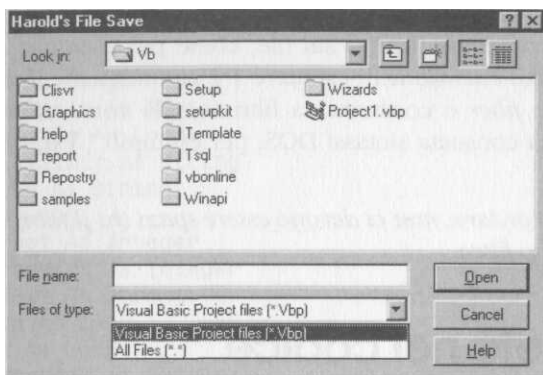
Figura 7.5 *si può utilizzare il dialogo PropertyPages del controllo dei dialoghi comuni per modificare le proprietà impostate il controllo.*

Si noti che la proprietà .Flags, che va inserita nel dialogo *Property Pages* con il suo valore numerico, assume ancora una volta il valore 2066, che ha il seguente significato:

```
cdlOFNOverWritePrompt + cdlOFNHelpButton + cdlOFNPathMustExist
```

Per aprire il dialogo con le proprietà e i flag che sono stati inseriti occorre una riga di codice:

```
CommonDialogl.ShowOpen
```

Si apre una finestra di dialogo comune, configurata nel modo stabilito dalle proprietà e dai flag impostati (si veda la Figura 7.6).

Figura 7.6
Il titolo Harold's File Save di questo dialogo è stato definito in fase di progettazione utilizzando la proprietà DialogTitle del controllo dei dialoghi comuni.

La proprietà Filter

La proprietà .Filter stabilisce il contenuto della casella di riepilogo *Files ofType* che compare nella parte inferiore dei dialoghi *File Save As* e *File Open.* Questa proprietà costituisce un elemento imFortante nella funzionalità di questo genere di dialoghi, e la sua sintassi può risultare particolarmente elaborata. Ecco come funziona. Si è già osservato in Figura 7.5 che l'inserimento della seguente proprietà .Filter nel visualizzatore

```
Visual Basic project files (*.Vbp)|*.vbp|All Files (*.*)|*.*
```

produce la casella di riepilogo *Files of Type* mostrata in Figura 7.6. Ovviamente questo può essere definito in fase di codifica come stringa di caratteri, e produce lo stesso risultato:

```
Dim Fstring as string

Fstring = _
"Visual Basic project files (*.Vbp)|*.vbpjAll Files (*.*)|*.*"
CommonDialogl.filter = Fstring
```

Supponiamo, a titolo di prova, di aggiungere un carattere vuoto alla fine della stringa Fstring:

```
Fstring = _
"Visual Basic project files (*.Vbp)|*.vbpjAll Files (*.*)|*.*  "
```

Anche se l'indicazione relativa a *AH Files (*. *)* compare nell'elenco di *Files of Type,* quando si seleziona questa opzione nessun file viene visualizzato. Cosa sta succedendo?
La sintassi della stringa equivalente alla proprietà .Filter, e viceversa, è la seguente:

Desd | *filter* | Desc2 | *filter* |...DescN | *filter*

Le descrizioni *Desc* possono essere indicate a piacere; non devono includere parentesi convenzionali per contenere il filtro sui file, come per esempio *All Files (*.*)* oppure *File di Frodo senza estensione.* Il carattere | è naturalmente il simbolo pipe, ASCIl 124. L'indicazione *filter è* costituita da filtri sui file con caratteri |olly, che vanno inseriti secondo la consueta sintassi DOS; per esempio *.Txt, *.Doc, *.Frm e così via.

Questo è un punto da ricordare: non ci devono essere spazi tra il filtro e il carattere pipe, oppure dopo l'ultimofiltro.

Flag e proprietà nel codice

Per quanto possa essere divertente programmare in modo visivo con Visual Basic, qualcuno preferisce stabilire con precisione i dettagli del lavoro di programmazione. Gli affezionati delle righe di codice sono in grado di definire tutte le proprietà e i flag dei dialoghi comuni in fase di codifica anziché utilizzare la finestra *Properties.* Il vantaggio fondamentale di questo tipo di procedura riguarda il fatto che si possono programmare dinamicamente i dialoghi comuni in modo che il loro aspetto e le caratteristiche possano essere modificate durante l'esecuzione. Se si definiscono i dialoghi comuni mediante righe di codice, un certo dialogo può occuparsi di diversi elementi, una prima volta con un determinato nome di file predefinito e la volta successiva con un nome differente. Uno stesso controllo dei dialoghi comuni può essere aperto in una qualsiasi delle modalità a disposizione.

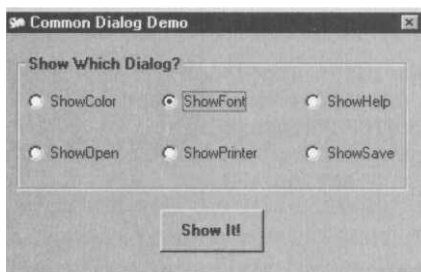
Una possibilità interessante è data dalla creazione difunzioni involucro personali, anche relative a server OLE che comprendano le vostre scelte dipartenza dei dialoghi comuni ma consentano dipassare le variabili da definire di volta in volta, come il nome difilepredefinito. (Si veda il Capitolo 10per un esempio di questo tipo condotto passo perpasso.)

Un'altra possibilità, se ve la sentite di lavorare direttamente con le API di Windows, consiste nel tralasciare Comdlg32.Ocx e utilizzare le funzioni che fanno parte di Comdlg32.Dll. In questo modo si hanno diversi vantaggi, tra i quali quello di dover distribuire un file in meno nel runtime (il controllo dei dialoghi comuni). Tuttavia, la semplice programmazione può risultare abbastanza complessa e coinvolge l'impiego del tipo OPENFILENAME. Anche se non è impossibile da fare, ho voluto esForre di seguito la definizione di tipo nel Listato 7.1, per darvi un'idea delle difficoltà che si possono incontrare:

Listato 7.1 *Definizione di tipo OPENFILENAME.*

```
Type OPENFILENAME
   lStructSize As Long
   hwndOwner As Long
   hInstance As Long
   lpstrFilter As String
   lpstrCustomFilter As String
   nMaxCustFilter As Long
   nFilterIndex As Long
   lpstrFile As String
   nMaxFile As Long
   lpstrFileTitle As String
   nMaxFileTitle As Long
   lpstrInitialDir As String
   lpstrTitle As String
   flags As Long
   nFileOffset As Integer
   nFileExtension As Integer
   lpstrDefExt As String
   lCustData As Long
   lpfnHook As Long
   lpTemplateName As String
End Type
```

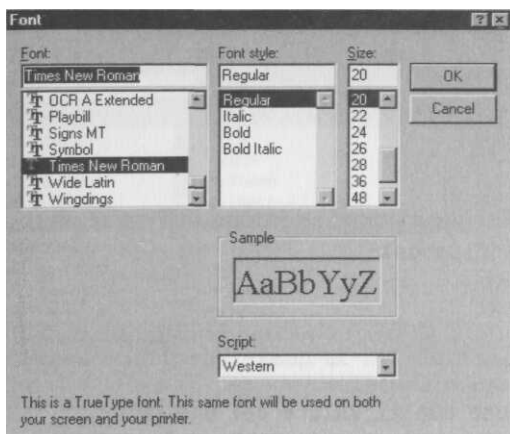
Nel CD-ROM allegato al libro è presente un semplice progetto, memorizzato come Common.Vbp, che utilizza un array di pulsanti opzione (si veda la Figura 7.7) per aprire dinamicamente tutti i dialoghi consentiti da un controllo dei dialoghi comuni.

Figura 7.7 *Il progmmma dimostrativo incluso nel CD-ROM illustra dinamicamente tutti i possibili dialoghi comuni.*

La codifica del programma utilizza la quinta lettera della didascalia relativa a ciascun pulsante di opzione *(ShowColor, ShowFont,* eccetera) per attivare il corretto valore della proprietà e l'istruzione *Show* corrispondente. Ho tentato anche di aggiungere ai dialoghi flag e valori delle proprietà il più possibile realistici; il codice è mostrato nel Listato 7.2.

Listato 7.2 *Apertura di dialoghi generici.*

```
Private Sub cmdShow_Click()
   Dim x As Integer, Which As Integer
   Which = 0
   For x = 0 To 5
      If optWhich(x).Value = True Then Which = x
   Next x
   Select Case UCase(Mid(optWhich(Which).Caption, 5, 1))
      Case "C"
         CommonDialog1.Flags = cdlCCFullOpen
         CommonDialog1.ShowColor   'Action = 3
      Case "F"
         CommonDialog1.Flags = cdlCFBoth + cdlCFTTOnly
         CommonDialog1.FontName = "Times New Roman"
         CommonDialog1.FontSize = 20
         CommonDialog1.ShowFont    'Action = 4
      Case "H"
         CommonDialog1.HelpFile = "VbS.Hlp"
         CommonDialog1.HelpCommand = cdlHelpContents
         CommonDialog1.ShowHelp   'Action = 6
      Case "O"
         CommonDialog1.DialogTitle = "Harold's File Open"
         CommonDialog1.Filter = _
      "Visual Basic project files (*.Vbp)|*.vbp|All Files (*.*)|*.*"
         CommonDialog1.Flags = cdlOFNAllowMultiselect +
            cdlOFNExplorer + cdlOFNLongNames + cdlOFNFileMustExist
         CommonDialog1.ShowOpen  'Action = 1
      Case "P"
         CommonDialog1.Flags = cdlPDHidePrintToFile
         CommonDialog1.ShowPrinter 'Action = 5
      Case "S"
         CommonDialog1.DialogTitle = "Harold's File Save"
         CommonDialog1.Filter =
         "Files will rule the world!|*.vbp|I don't think so!|*.*"
         CommonDialog1.Flags = cdlOFNOverwritePrompt + _
            cdlOFNHelpButton + cdlOFNPathMustExist
            'CommonDialog1.Flags = 2066
         CommonDialog1.ShowSave    'Action =2
      Case Else
         MsgBox "Whoops!"
    End Select
End Sub
```

Quando si esegue il progetto dimostrativo, è possibile aprire tutti i tipi di dialogo. La Figura 7.8 mostra il dialogo *Font* così come viene definito da questo programma.

Figura 7.8 *Questo è ilgenerico dialogo Font, con flag assegnati in modo da rendere disponibili i font TrueType relativi allo schermo e alla stampante.*

Altre informazioni sulla guida

Dovrebbe essere noto che i file definiti con la proprietà HelpFile *dei dialoghi comuni sono "vecchio stile" (al contrario dei file di guida in formato HTML). Per avere maggiori informazioni su come si lavora con i file di guida HTML, si veda il Capitolo 34.*

La proprietà HelpCommand del controllo dei dialoghi comuni può essere utilizzata anche per stabilire come viene visualizzato un file di guida. Si può trovare un elenco completo delle costanti principali da utilizzare per questa finalità in Object Browser selezionando MsComDlg, in corrispondenza di HelpConstants. Per esempio, se HelpCommand include cdlHelpForceFile, il file di guida viene visualizzato con un solo font; inoltre, se HelpCommand include la costante predefinita cdlHelpIndex, viene visualizzato l'indice del file.

Rilevare il comando Cancel

Quando si programmano i dialoghi comuni, di solito si vuole stabilire se l'utente interrompe il dialogo premendo il pulsante *Annulla (Cancel).* In questo modo l'utente decide di non proseguire l'azione prevista dal dialogo e i nuovi valori appena definiti devono essere tutti ignorati. (A questo proposito, per tutti i dialoghi a eccezione di *ShowHelp,* chiudere il dialogo premendo il pulsante di chiusura visibile nella barra del titolo equivale a premere il pulsante *Cancel.)*
Vediamo come rilevare la richiesta di annullamento nel programma d'esempio. Ancora prima che un dialogo qualsiasi venga aperto occorre definire la proprietà .CancelError dei dialoghi comuni con il valore True; si può fare questo nella finestra *Properties* oppure in fase di codifica. Se .CancelError è definito True, la cancellazione da parte dell'utente genera un errore, cdlCancel (&H7FF3&).

Si inserisce quindi una routine di gestione dell'errore dopo l'istruzione CancelError:

```
CommonDialog!.CancelError = True
On Error GoTo ErrHandler
```

Infine, si aggiunge la codifica che riForta la gestione dell'errore alle condizioni iniziali e si occupa dell'errore cdlCancel:

```
    End Select
    On Error GoTo 0 'Reinizializza la questione degli errori
     'L'utente non ha premuto Cancel.
     'Intraprendi un'azione con il valore del dialogo.
    Exit Sub
ErrHandler:
   If Err = cdlCancel Then
      MsgBox "L'utente ha premuto Cancel - ignora i valori del dialogo!"
   End If
End Sub
```

L'istruzione

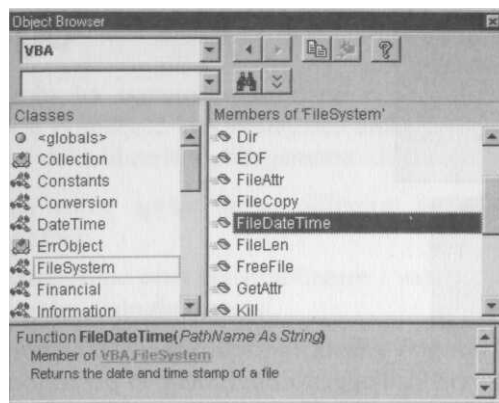
```
On Error GoTo 0
```

viene raggiunta dopo un utilizzo riuscito di un dialogo comune e disattiva la gestione degli errori prevista dalla procedura corrente.
A questo proposito, si può utilizzare ErrHandler per rilevare altri errori nei dialoghi comuni, oltre a cdlCancel. Si può ottenere una lista completa delle costanti relative agli errori dei dialoghi comuni utilizzando Object Browser (si vedano i membri della classe ErrorConstants della libreria MSComDlg) oppure si consulti la guida in linea di VB alla voce CommonDialog Error Constants.

Oggetti di FileSystem

In Visual Basic 6 è possibile utilizzare i membri dell'oggetto VBA FileSystem *per elaborare file, directory e sistemi con lo stesso livello di astrazione disponibile per l'involucro dei dialoghi generici. Per avere informazioni sui membri di* FileSystem, *aprire Object Browser (premendo F2 oppure selezionandolo dal menu* View); *i membri dell'oggetto* FileSystem *si trovano nella libreria VBA, come mostrato in Figura 7.9.*

Il Listato 7.3 illustra come si possono visualizzare data e ora relativi a un file selezionato mediante il controllo dei dialoghi comuni.

Figura 7.9 *È possibile utilizzare Object Browser per visualizzare i membri dell'oggetto VBA FileSystem.*

Listato 7.3 *Utilizzo dell'oggetto FileSystem.*

```
Private Sub cmdShow_Click()
   CommonDialog1 .CancelError = True
   On Error GoTo ErrHandler
   CommonDialog1.DialogTitle = "Harold's File Open"
   CommonDialog1.Filter =
   "Visual Basic project files (*.Vbp)|*.vbp|Visual Basic group
      files (*.Vbg)|*.vbg|All Files (*.*)|*.*"
   CommonDialog1.Flags = cdlOFNAllowMultiselect +
      cdlOFNExplorer + cdlOFNLongNames + cdlOFNFileMustExist
   CommonDialog1.ShowOpen  'Action = 1
   On Error GoTo 0  'Reinizializza la questione deg1i errori
   'L'utente non ha annullato. Chiama FileDateTime
   MsgBox FileDateTime(CommonDialog1.FileName), ,
      "File Date/Time Stamp"
   Exit Sub
ErrHandler:
   If Err = cdlCancel Then
      MsgBox "L'utente ha premuto Cancel - ignora i valori del dialogo!"
   End If
End Sub
```

La Figura 7.10 mostra l'utilizzo di questa routine per visualizzare data e ora di un file che risulta creato nell'anno 2000.

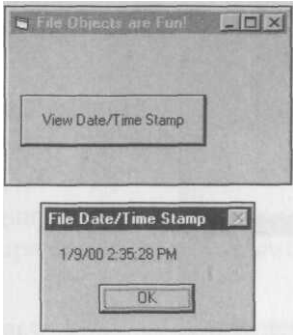
Figura 7.10
La proprietà FileDateTime dell'oggetto FileSystem consente di visualizzare data e ora di un particolare file.

La Tabella 7.2 contiene i membri dell'oggetto *FileSystem;* si possono trovare maggiori informazioni sulla manipolazione dei file nel Capitolo 16.

Tabella 7.2 *Membri dell'oggetto FileSystem.*

Membro	**Finalità**
ChDir	Modifica la directory corrente oppure quella predefinita
ChDrive	Modifica il drive corrente
CurDir	Restituisce il percorso corrente
Dir	Restituisce il nome di un file, di una directory o di una cartella abbinati al membro indicato
EOF	Restituisce un valore booleano che indica se è stata raggiunta o meno la fine di un file
FileAttr	Restituisce la modalità per i file aperti con l'istruzione Operi
FileCopy	Copia un file
FileDateTime	Restituisce data e ora
FileLen	Restituisce la lunghezza di un file in byte
FreeFile	Restituisce il prossimo numero disponibile nell'utilizzo dell'istruzione Open
GetAttr	Restituisce gli attributi di un file, di una directory o di una cartella
Kill	Cancella un file
Loc	Restituisce la posizione all'interno di un file aperto
LOF	Restituisce la dimensione in byte di un file aperto
MkDir	Crea una directory
Reset	Chiude tutti i file aperti
RmDir	Cancella una directory
Seeks	Definisce oppure ritorna la posizione corrente all'interno di un file aperto
SetAttr	Definisce le informazioni sugli attributi

Riepilogo

Questo capitolo ha trattato la versione 6 del controllo dei dialoghi comuni, Comdlg32.Ocx. Comdlg32.Ocx costituisce una comoda passerella tra Visual Basic e molte funzioni della libreria a collegamento dinamico di Comdlg32.Dll.

- Si è visto come aprire tutti i differenti tipi di dialoghi disponibili in Comdlg32.Ocx.
- Ho spiegato come ottenere e utilizzare i valori corretti delle costanti relative ai flag dei dialoghi comuni.
- Avete appreso come definire proprietà e flag mediante la finestra *Properties.*
- Ho esaminato la definizione della codifica relativa a proprietà e flag.
- Avete visto come rilevare la cancellazione di un dialogo.
- Avete imparato a utilizzare l'oggetto `FileSystem` di VBA.
- Ho esaminato alcuni trucchi da utilizzare con i dialoghi comuni, tra i quali il valore `.Flag` richiesto dal dialogo *Font* per sapere se il proprio sistema dispone di font.
- Ho anche analizzato il trucco per definire correttamente le proprietà `.Filter`.

CONTROLLI D'INTERFACCIA UTENTE

- Apprendimento delle caratteristiche di programmazione mediante il loro impiego
- Inserimento dei controlli dell'interfaccia utente nella Toolbox
- Creazione di fogli proprietà
- Utilizzo del controllo TabStrip
- Creazione di wizard e analisi del codice dei wizard
- Utilizzo delle demo ProgressBar e Slider
- Utilizzo della demo editor di testo
- Utilizzo del controllo Coolbar
- Utilizzo del controllo FlatScrollBar
- Visualizzazione delle gerarchie mediante i controlli ListView e TreeView
- Utilizzo dei controlli calendario
- Creazione di un controllo Spinner (selettore)
- Utilizzo del controllo SysInfo
- Utilizzo del controllo MSFlexGrid
- Utilizzo del controllo ImageCombo

Le versioni Professional e Enterprise Edition di Visual Basic 6 sono distribuite con una serie di controlli particolari che sono stati progettati per dare ai programmi l'aspetto e le capacità della shell. I programmi che costruite utilizzando questi controlli sono destinati a funzionare nel modo in cui gli utenti si aspettano che lavori l'interfaccia utente.

Molti dei controlli distribuiti con VS6 sono completamente nuovi; inoltre, molti dei vecchi controlli sono stati ampiamente migliorati.

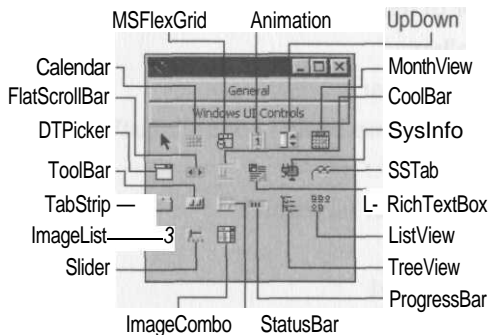
I controlli trattati in questo capitolo sono i seguenti:

- CoolBar
- DTPicker
- FlatScrollBar

- ImageCombo
- ImageList
- ListView
- MonthView
- MSFlexGrid
- ProgressBar
- RichTextBox
- Slider
- SSTab
- StatusBar
- SysInfo
- TabStrip
- ToolBar
- TreeView
- UpDown

Molti controlli dell'interfaccia utente precedente sono rappresentati da icone nella Toolbox, come mostrato in Figura 8.1.

Figura 8.1

Queste icone della Toolbox rapresentano gran parte dei controlli dell'interfaccia utente di Windows.

Questo capitolo illustra come si utilizzano questi controlli per creare applicazioni che non solo rispettano le convenzioni adottate dall'interfaccia utente di Windows, ma sono in stile Windows all'ennesima potenza!

Provare per credere

Sono convinto che sia sempre meglio toccare con mano piuttosto che perdersi in chiacchiere; per esperienza personale credo che il modo migliore per imparare la programmazione consista nel provare e nel vedere come funziona. In questa ottica, il presente capitolo focalizza l'attenzione su progetti di esempio che:

- mostrano come si utilizza il controllo SSTab per creare un foglio proprietà;

- spiegano come creare un wizard in VB, utilizzando un modello generico; la codifica per il modello di wizard è disponibile nel CD-ROM allegato al libro con il nome di Wizard.Vbp, nella directory relativa al Capitolo 8;
- illustrano il funzionamento di ProgressBar e di Slider;
- mostrano quello che si può fare con RichTextBox;
- mostrano l'impiego dei controlli TreeView, ListView e ImageList;
- mostrano come si utilizza il controllo UpDown, in combinazione con una casella di testo, per creare uno spinner;
- illustrano l'utilizzo dei nuovi controlli di VB6, inclusi CoolBar, FlatScrol-IBar, Calendar, MonthView, DTPicker e ImageCombo.

Inserimento dei controlli dell'interfaccia utente nella Toolbox

Prima di iniziare dovete essere sicuri che i controlli che volete utilizzare siano presenti nella vostra Toolbox di Visual Basic. Se i controlli dell'interfaccia utente di Windows non sono ancora presenti nella Toolbox, bisogna aggiungerli utilizzando il dialogo *Componente* (che si trova nel menu *Project* oppure facendo clic destro nella Toolbox). La Tabella 8.1 mostra il nome del controllo che compare nel dialogo *Components,* il nome del file corrispondente e i controlli ActiveX contenuti nella voce *Component* del dialogo. (Un singolo file .Ocx può contenere parecchi controlli.)

Tabella 8.1 *Nomi dei componenti e controlli.*

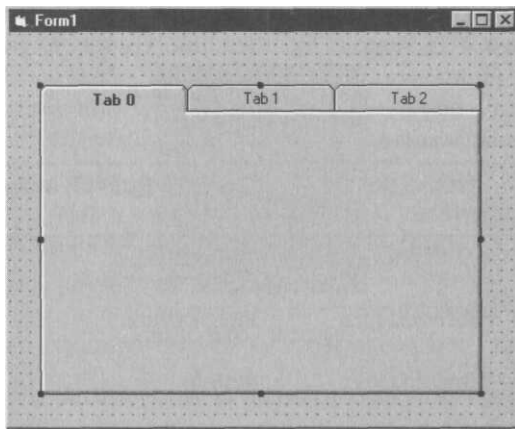
Nome che compare nel dialogo Components	Nome del file	Controlli ActiveX inclusi
Microsoft FlexGrid Control 6.0	Msflxgrd.Ocx	MSFlexGrid
Microsoft Rich Textbox Control 6.0	Richtx32.Ocx	RichTextBox
Microsoft SysInfo Control 6.0	Sysinfo.Ocx	SysInfo
Microsoft Tabbed Dialog Control 6.0	Tabct!32.Ocx	SSTab
Microsoft Windows Common Controls 6.0	Mscomctl.Ocx	ImageCombo, ImageList, ListView, ProgressBar, Slider, StatusBar, TabStrip, ToolBar, TreeView
Microsoft Windows Common Controls-2 6.0	Mscomct2.Ocx	Animation, DTPicker, FlatScrollBar, MonthView, UpDown
Microsoft Windows Common Controls-3 6.0	Comct332.Ocx	CoolBar

È facile aggiungere i controlli dell'interfaccia utente di Windows a una scheda di Toolbox, come mostrato in Figura 8.1. In primo luogo fate clic destro sulla Toolbox e selezionate Add Tab *nel menu di scelta rapida che compare. Quindi definite un nome appropriato per la nuova scheda, per esempio "Windows UT Controls". Fate clic su OK. Infine trascinate i controlli che volete avere nella nuova scheda, oppure aggiungeteli utilizzando il dialogo* Component. *Per essere sicuri che la Toolbox ricordi questa nuova definizione nelle sessioni di lavoro successive, salvate un progetto con questa configurazione come modello (in altre parole, salvate il progetto nella directory Template\Projects). La prossima volta che desiderate avere la Toolbox con la configurazione personalizzata che avete appena creato, è sufficiente aprire il progetto Tempiale.*

Creazione di un foglio proprietà

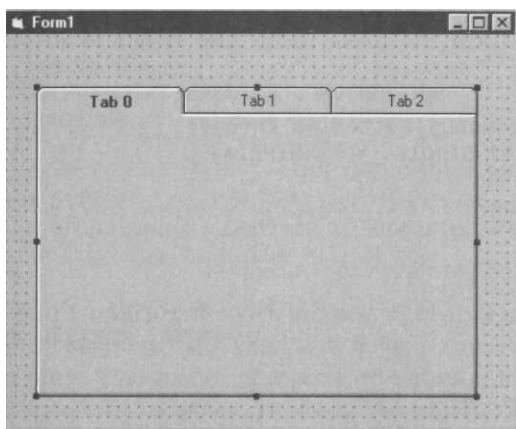
I fogli proprietà possono essere creati facilmente utilizzando i controlli a schede.

Una volta avviato un nuovo progetto, è sufficiente inserire un controllo SSTab *nel modulo. In questa situazione di partenza il dialogo a schede risulta simile a quello mostrato in Figura 8.2. (Questa semplice applicazione è disponibile nel CD-ROM allegato al libro con il nome di Sheet. Vbp.)*

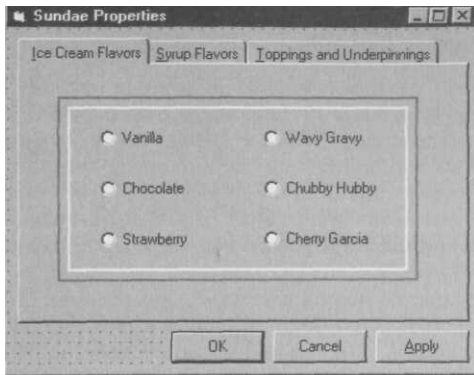
Figura 8.2
Un controllo SSTab trascinato sopra un form.

Se si confronta questo dialogo con un tipico dialogo *Properties* di Windows (si veda la Figura 8.3), si può notare che la forma della scheda non è esattamente uguale a quella presente nel dialogo *Properties* (anche se le differenze sono minime). Per rimediare a questa situazione ho modificato la proprietà Style predefinita relativa al controllo SSTab (la pagina delle proprietà che va utilizzata per questo è mostrata in Figura 8.3). La proprietà *Style* predefinita, 0-ssStyleTabbedDialog, deve essere cambiata in 1-ssStylePropertyPage.

La storia dei due controlli a schede

Avete probabilmente notato che le versioni Professional e Enterprise Edition di VB6 sono distribuite con due controlli a scheda: SSTab e TabStrip. A dispetto delle lettere iniziali del suo nome, SSTab non ha niente a che fare con le truppe d'assalto di nefasta memoria, si tratta più semplicemente di un controllo a schede creato originariamente dalla Sheridan Software. Molti sviluppatori preferiscono questo controllo rispetto a TabStrip, la scheda originale di Microsoft che viene distribuita come parte dell'insieme di controlli di Windows.
Entrambi i controlli sono estremamente potenti e flessibili; quale usare è un problema legato solo alle preferenze personali. (Uno degli argomenti a favore dell'impiego di TabStrip rispetto a SSTab riguarda il fatto che se il vostro progetto utilizza già componenti inclusi in Commctl32.Ocx, nella vostra applicazione non si deve distribuire un file aggiuntivo.)
Il foglio proprietà dell'esempio trattato utilizza SSTab; vedremo più avanti in questo stesso capitolo come si utilizza il controllo TabStrip.

Figura *8.3*
La pagina delle proprietà di SSTab costituisce un tipico dialogo delle Properties di Windows (si noti che è possibile utilizzare questo dialogo per configurare l'aspetto runtime del controllo SSTab).

Nelle versioni più vecchie di Visual Basic i nomi delle costanti relative a queste impostazioni di stile erano differenti; venivano utilizzate rispettivamente 0-Microsoft Office Tabbed Dialog *e* 1-Windows '95 Property Page.

Nel caso di questa semplice applicazione ho avuto bisogno di tre sole schede, per cui andava bene l'impostazione predefinita per la proprietà TabCount; ho poi inserito la didascalia relativa alla prima scheda nella casella di testo TabCaption. Successivamente mi sono spostato sulla scheda seguente facendo clic sulla freccia destra visibile sulla destra della casella di testo TabCaption.
Una volta definite le didascalie per le altre due schede, mi sono dedicato alla sistemazione dei pulsanti di comando sul fondo del form VB. Sempre con riferimento al tipico dialogo in stile Windows (Figura 8.3) ho visto che servono tre pulsanti: *OK, Cancel* e *Apply.* Se si utilizza il dialogo *Properties* di Windows, si può notare che il pulsante *Apply* viene attivato solo quando l'utente effettua una modifica qualsiasi. Il dialogo *Properties* da me realizzato con tutte queste caratteristiche è mostrato in Figura 8.4.

Figura 8.4 La simulazione completa in Visual Basic del dialogo Properties di Windows

Vediamo come funziona l'applicazione. Quando viene lanciato il programma, compare sullo schermo un form che presenta l'immagine bitmap di una coppa gelato. Per modificare le proprietà della coppa gelato, l'utente può selezionare la voce Properties dal menu Sundae in cima al form oppure può fare clic destro sull'immagine ed aprire un menu popup (noto come menu di scelta rapida). La codifica che genera il menu popup utilizza lo stesso menu che compare in cima al form. Vediamo come è fatto:

```
Private Sub imgSundae_MouseUp(Button As Integer,
   Shift As Integer, X As Single, Y As Single)
   If Button = 2 Then
      PopupMenu mnuSundae
   End If
End Sub
```

Poi compare sullo schermo il dialogo *Sundae Properties;* l'utente può effettuare le selezioni desiderate su ciascuna pagina a schede. La caratteristica ingegnosa di questa applicazione è che il pulsante opzione selezionato dall'utente su una determinata pagina non rimane definito se l'utente si sposta su una pagina diversa. In sostanza, nel caso in cui l'utente si sposta in avanti o all'indietro tra le schede, ho inserito una routine chiamata SetPreviousTab che ripristina il valore precedente nella scheda del controllo. Il Listato 8.1 mostra come si crea un foglio proprietà.

Listato 8.1 *Creazione di un foglio proprietà.*

```
Private Sub tabProperty_Click(PreviousTab As Integer)
    Dim X As Integer
    Select Case PreviousTab
    Case 0
      For X = 0 To 5
         If optIceCream(X) Then
            IceCream = optIceCream(X).Caption
      Next X
   Case 1
      For X = 0 To 5
```

```
         If optSyrup(X) Then _
            Syrup = optSyrup(X).Caption
      Next X
   Case 2
      Por X = 0 To 5
         If optToppings(X) Then
            Toppings = optToppings(X).Caption
      Next X
   Case Else
      MsgBox "Non ho così tante schede, amico!"
End Select
SetThisTab
End Sub

Private Sub SetThisTab()
   Dim X As Integer
   Select Case tabProperty.Tab
   Case 0
      Por X = 0 To 5
         If optIceCream(X).Caption = IceCream Then
            optIceCream(X).Value = True
      Next X
   Case 1
      Por X = 0 To 5
         If optSyrup(X).Caption = Syrup Then _
            optSyrup(X).Value = True
      Next X
   Case 2
      Por X = 0 To 5
         If optToppings(X).Caption = Toppings Then
            optToppings(X).Value = True
      Next X
   Case Else
      MsgBox "Errore interno!"
End Select
End Sub
```

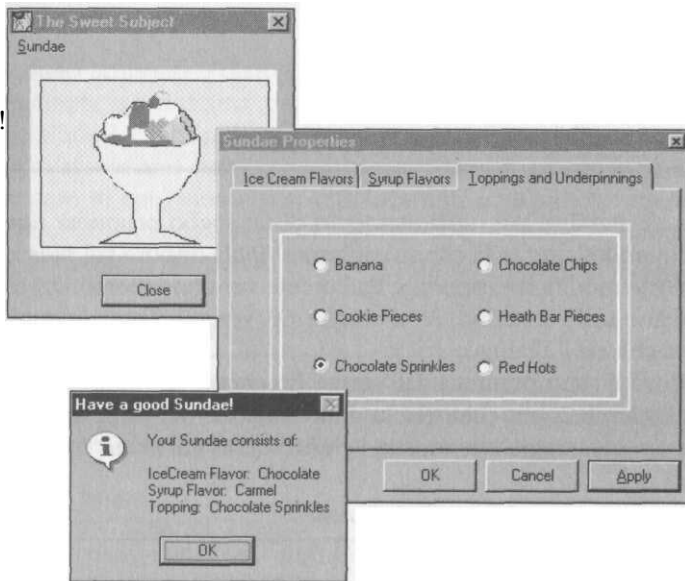
L'ultima cosa da stabilire nel funzionamento di un foglio proprietà riguarda quello che succede quando l'utente fa clic sul pulsante *Apply* oppure sul pulsante *OK.* Se si fa clic su *Apply* le modifiche apportate dall'utente vengono memorizzate e il dialogo *Properties* rimane sullo schermo. Se si sceglie *OK* vengono ugualmente registrate le modifiche e si chiude il dialogo.
Per stabilire cosa è stato premuto dall'utente ho creato la routine GetUserChoices, mostrata nel Listato 8.2; per chiudere la dimostrazione del programma ho deciso di far comparire un messaggio che mostra l'elenco delle selezioni "memorizzate".

Listato 8.2 *Determinazione delle scelte dell'utente.*

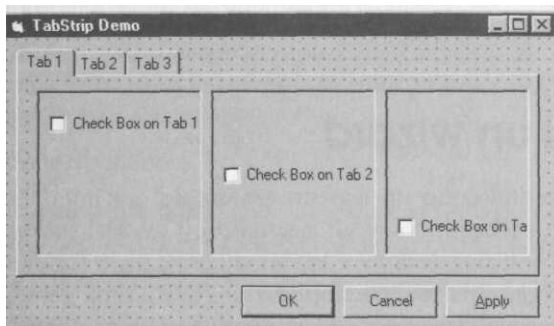
```
Private Sub GetUserChoices()
   Dim X As Integer, Msg As String
   Select Case tabProperty.Tab
      Case 0
```

```
        For X = 0 To 5
          If optIceCream(X) Then IceCream = optIceCream(X).Caption
        Next X
     Case 1
        Fon X = 0 To 5
          If optSyrup(X) Then Syrup = optSyrup(X).Caption
        Next X
     Case 2
        For X = 0 To 5
          If optToppings(X) Then Toppings = optToppings(X).Caption
        Next X
     Case Else
        MsgBox "Non ho così tante schede amico!"
   End Select

   If IceCream = "" And Syrup = "" And Toppings = "" Then
      Msg = "Non hai scelto nulla!"
   Else
      Msg = "Your Sundae consists of:" + vbCrLf + vbCrLf
      Msg = Msg + "IceCream Flavor:  " + IceCream + vbCrLf
      Msg = Msg + "Syrup Flavor:  " + Syrup + vbCrLf
      Msg = Msg + "Topping:  " + Toppings
   End If
      MsgBox Msg, vbInformation, "Buon appetito!"
End Sub
```

Ecco fatto! Quando si esegue l'applicazione l'utente può sbizzarrirsi a creare una coppa di suo gusto, come mostrato in Figura *8.5*.

Figura 8.5
Questa coppa gelato è un vero attentato per i nostri denti!

Utilizzo del controllo TabStrip

Tra i controlli comuni di Microsoft Windows (in altre parole, che fanno parte del file Comctl32.Ocx) è compreso il controllo TabStrip; questo controllo funziona in modo diverso dal controllo SSTab ed è mia opinione che non sia altrettanto intuitivo, in quanto è richiesto del codice per farlo funzionare. Invece di accedere alle singole pagine a schede in fase di progettazione, come nel caso del controllo SSTab, si devono elaborare le pagine TabStrip mediante codice in fase di esecuzione. Questo significa che occorre sistemare tutti i contenitori del controllo nella prima pagina a schede del controllo TabStrip (si veda la Figura 8.6). A questo punto, mediante codice si possono dimensionare i contenitori in modo che si adattino all'area client della pagina a schede e si Forta il contenitore appropriato in primo piano utilizzando il metodo ZOrder.

Figura 8.6 È stato messo un controllo TabStrip in un forni e tre caselle di immagine vengono utilizzate come contenitori per altri controlli.

Un esempio di controllo TabStrip funzionante con contenitori di immagini è disponibile sul CD-ROM allegato al libro con il nome di TabStrip. Vbp.

Vediamo un esempio di come si lavora con il controllo TabStrip. Si avvia un nuovo progetto e si aggiunge un TabStrip a un form, dimensionandolo a piacere. Si apre la finestra *TabStrip Control Properties* che si trova sotto la proprietà *Custom* e ci si sposta sulla pagina *Tabs* per definire tre pagine a schede. Sulla pagina *Tabs* si inserisce una didascalia per la prima pagina a schede, quindi si preme il pulsante *Inserì Tab* per creare una seconda pagina. Si inserisce una didascalia per questa pagina, quindi si crea una terza pagina premendo di nuovo il pulsante *Inserì Tab.* Si inserisce la didascalia per questa ultima pagina, quindi si fa clic su *OK* per chiudere la finestra *Properties.*
Si inserisce una matrice di tre controlli casella di immagine, che agiscono da contenitore del controllo TabStrip, come mostrato sempre in Figura 8.6. Si aggiunge una differente casella di controllo per ciascun riquadro, quindi si apre la finestra *Code* del form. Si aggiunge il codice seguente all'evento Load del form per dimensionare la matrice di controlli immagine, in modo da ricoprire l'area client del controllo a scheda e per fissare il primo controllo immagine in cima:

```
Private Sub Form_Load()
   Dim i As Integer
   Fori=OTo2
   picContainer(i).Left = tabDemo.ClientLeft
```

```
    picContainer(i).Top = tabDemo.ClientTop
    picContainer(i).Height=tabDemo.ClientHeight
    picContainer(i).Width = tabDemo.ClientWidth
Next
    picContainer(0).ZOrder 0
End Sub
```

Poi si inserisce una riga di codice in corrispondenza dell'evento Click di TabStrip per portare in primo piano il corretto controllo immagine quando l'utente cambia le pagine a schede.

```
Private Sub tabSundae_Click()
    picContainer(tabDemo.SelectedItem.Index - 1).ZOrder 0
End Sub
```

Se si esegue a questo punto l'applicazione si può notare come le caselle immagine vengono ridimensionate e posizionate correttamente in primo piano quando viene selezionata una specifica pagina a schede.

Creazione di un wizard

È noto che i wizard costituiscono un aspetto essenziale dell'interfaccia utente di Windows; a dire il vero i wizard, come gli assistenti e i tutorial (nomi utilizzati da società diverse da Microsoft per designare la stessa cosa), sono fondamentali per il funzionamento della maggior parte delle applicazioni dell'ultima generazione.
In Windows i wizard sono utilizzati per guidare l'utente nello svolgimento di attività complesse. I wizard appaiono all'utente come una serie di pannelli di dimensioni e aspetto identici tra loro, ciascuno dei quali prevede i pulsanti Back, Next e Cancel. (Il pulsante Back è disattivo in corrispondenza del primo pannello della serie.)
Ciascun pannello contiene anche tutti i controlli necessari perché l'utente possa portare a termine il compito previsto da quel particolare pannello. Quando si raggiunge l'ultimo pannello la didascalia del pulsante Next si modifica in Finish; se l'utente fa clic su questo pulsante, viene eseguito il codice necessario per completare l'attività prevista dal wizard. Un esempio di wizard che fa parte del sistema operativo di Windows 95/98 è costituito da Add New Hardware Wizard (.Nuovo hardware: questa procedura guidata si trova nel Pannello di controllo).
Alcuni wizard, come per esempio ChartWizard di Excel, funzionano in modo leggermente diverso in quanto è sempre disponibile il pulsante Finish; questo consente all'utente di trascurare i pannelli successivi in qualunque fase dell'attività. Se si vuole che il proprio wizard si comporti in questo modo è sufficiente modificare il progetto campione.
Si può pensare che un wizard sia costituito da un certo numero di form differenti; visto in questo modo, un wizard può per esempio nascondere il primo form quando va a mostrare il secondo. Tutti i form devono presentare i controlli in comune (Back, Next e Cancel) esattamente nella stessa posizione.
Nulla vieta di predisporre un wizard secondo una logica su più form, anche se risulta più facile, efficiente ed elegante utilizzare un singolo form che presenti i con-

Utilizzo di un form di opzioni modello

Visual Basic 6 prevede diversi form che potete utilizzare come modello per i vostri form. Questi form sono salvati nella directory Template\Forms; vi si può accedere ogni volta che si vuole aggiungere un form all'interno di un progetto.
Uno dei form modello è costituito da un dialogo *Options, frmOption.* Questo form utilizza il controllo *TabStrip* ed è molto simile al dialogo visibile in Figura 8.6. Selezionare il form *Options* come modello rappresenta una scorciatoia che consente di mettere *TabStrip* al vostro servizio. Non vengono solo aggiunti i controlli necessari a *TabStnp,* ma viene anche sistemata al posto giusto parte del codice necessario, per esempio quella che consente all'utente di passare da un pannello al successivo.

trolli comuni. Si possono organizzare i diversi controlli in modo da creare l'illusione di più pannelli multipli, agendo sui controlli da rendere visibili e invisibili.

// *CD-ROM allegato al libro mette a disposizione il codice da utilizzare come modello per creare facilmente un wizard basato sulla presenza di un singolo form; i file necessari sono Wizard.Bas e Wizard.Frm.*

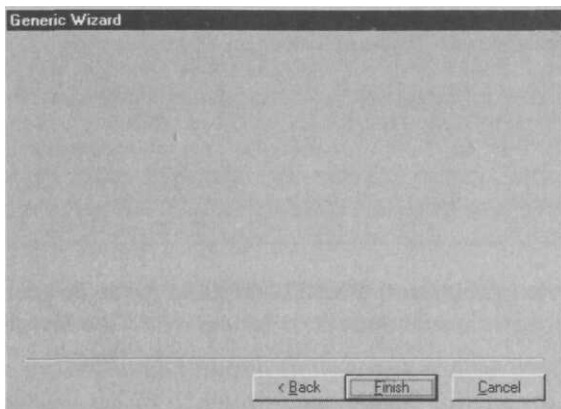
In questa sezione vediamo:

- Un'analisi passo per passo che illustra come utilizzare il generico modello di wizard e la codifica per creare facilmente i vostri wizard.
- Un wizard di esempio SillyWiz.Vbp da manipolare e sperimentare.
- Una spiegazione di come funziona il codice di un wizard generico.

Il wizard Wizard

Visual Basic 6 viene distribuito con un wizard Wizard, noto anche con il nome di VB 6 Wizard Manager, strumento progettato per aiutare nella creazione di wizard che costituiscano degli add-in. Si esegue il wizard Wizard selezionandolo dal menu *Add-Ins* di Visual Basic, a patto che sia prima stato caricato in Add-Ins Manager.
Scopo principale del wizard Wizard è quello di gestire l'interfaccia di progetto dei pannelli di un wizard, dove quest'ultimo va inteso come add-in che interagisce con istanze dell'ambiente di Visual Basic. Vedremo nel Capitolo 30 come si crea questo tipo di wizard, che può apparire ai vostri utenti come una parte a se stante dell'ambiente VB.
Si deve ricordare che il wizard Wizard produce una minima quantità di codifica di implementazione runtime oltre a quella richiesta per registrare come add-in i wizard creati in questo modo. Scopo principale del wizard Wizard è di occuparsi della visualizzazione del pannello di progettazione, per creare in modo visuale il wizard come add-in. Nulla vieta comunque di usare pannelli creati dal wizard Wizard come punto di partenza dei vostri wizard, anche se questi sono più simili a Sundae che al genere add-in. (Il wizard "Sundae" è descritto più avanti in questo capitolo; gli add-in vengono spiegati nel Capitolo 29.) Occorre ricordare che siete comunque responsabili dell'implementazione runtime dei vostri wizard, che siano o meno degli add-in.

Il wizard generico non è molto eccitante da guardare dato che si tratta semplicemente di un dialogo vuoto (si veda la Figura 8.7) anche se, lo vedremo fra un istante, è dotato di parecchie funzionalità.

Figura 8.7 *Si parte dal wizard generico per costruire wizard di nostro gusto.*

Creazione di wizard

Vediamo come si utilizza il wizard generico per creare nuovi wizard:

1. Copiare i file del wizard sul proprio disco fisso: Wizard.Bas, Wizard.Frm e Wizard.Vbp.
2. Aprire il progetto Wizard.Vbp. Salvare ciascun file, ed il progetto, con un nuovo nome.
3. Stabilire il numero di pannelli desiderato e il numero di controlli che si vogliono su ciascun pannello.
4. Aprire il modulo del codice Wizard.Bas in Code Editor. Individuare la procedura SetUpWiz, che dovrebbe essere simile alla seguente:

   ```
   Private Sub SetUpWiz()
   'Pannello 1
   ItemsInPanel0
   'Pannello 2
   ItemsInPanel0
   'Pannello 3
   ItemsInPanel0
   End Sub
   ```

   Il numero di chiamate a ItemsInPanel rappresenta il numero di pannelli nel wizard; l'argomento passato a ItemsInPanel dice al wizard quanti controlli ci sono su quel pannello. Si deve modificare la codifica in modo da adeguarla al proprio wizard; per esempio, se si vuole creare un wizard con quattro pannelli, con due controlli su ciascun pannello, si deve modificare SetUpWiz come segue:

```
Private Sub SetUpWiz()
    'Pannello 1
    ItemsInPanel 2
    'Pannello 2
    ItemsInPanel 2
    'Pannello 3
    ItemsInPanel 2
    'Pannello 4
    ItemsInPanel2
End Sub
```

5. A questo punto si aggiungono i controlli desiderati in Wizard.Frm. Il wizard gestisce la visibilità o meno di questi controlli mediante le proprietà .Tag, per cui è imFortante definirle con attenzione. Il primo controllo da aggiungere dovrebbe avere la proprietà .Tag definita come 5. (Questo valore è superiore di uno al numero dei controlli generici di un wizard. Sono quattro i controlli presenti su ogni pannello: tre pulsanti e una riga.) Il secondo controllo deve avere la proprietà .Tag definita come 6, e così via. Si devono definire le proprietà .Tag in modo che i pannelli possano essere gestiti correttamente. Nell'esempio con due controlli per pannello i controlli hanno le proprietà .Tag mostrate nella Tabella 8.2.

Tabella 8.2 *Proprietà .Tagper i controlli di un wizard a quattro pannelli con due controlli perpannello.*

Pannello	Proprietà .Tag del controllo
1	5,6
2	7,8
3	9,10
4	10,11

6. Questo per gestire l'aspetto dei pannelli. Si deve anche aggiungere la codifica eseguibile che faccia effettivamente lavorare il wizard. Di solito questa codifica è richiamata dopo che l'utente ha fatto clic sul pulsante *Finish.* Ho aggiunto una procedura di questo tipo, chiamata ShutDown, che prevede un messaggio che viene richiamato quando l'utente fa clic su *Finish:*

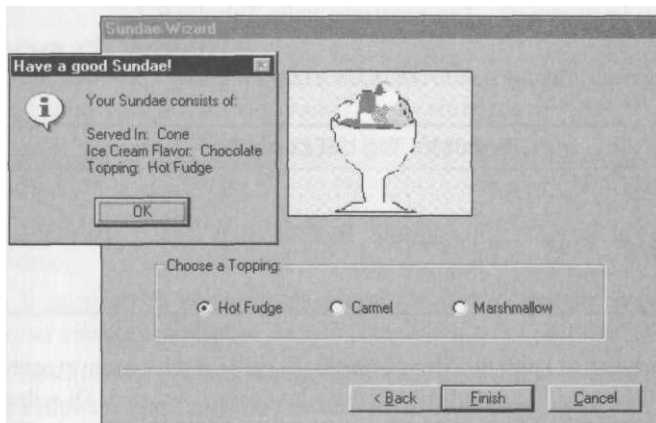
```
Public Sub ShutDown()
   MsgBox "Sto facendo quello che devo fare
   con i valori e il resto... sto finendo"
   Unload ThisWizard
   End
End Sub
```

Si deve ricordare che a questo punto si hanno ancora a disposizione i valori inseriti mediante i controlli del wizard; sono semplicemente nascosti, non sono stati scaricati.

Si può definire il codice eseguibile anche quando l'utente preme *Cancel,* per ripulire, ritornare alle condizioni iniziali o qualunque altra operazione. Questo codice di solito è inserito in corrispondenza dell'evento cmdCancel del modulo utilizzato come modello.
Si può aggiungere codice eseguibile agli eventi di un qualunque controllo dei pannelli (si ricordi che i controlli che non sono visibili vengono automaticamente disabilitati).
Si può anche aggiungere il codice da eseguire quando si va avanti e indietro tra i pannelli (si deve controllare probabilmente da quale pannello ci si è mossi) ma questa è una questione più complessa da risolvere.

Il wizard Sundae

La Figura 8.8 mostra il wizard Sundae che consente all'utente di mettere insieme una ricca coppa di gelato scegliendo tra diversi contenitori, gusti e decorazioni; questo wizard si trova nel CD-ROM allegato al libro con il nome SillyWiz ed è molto simile all'esempio relativo al foglio proprietà a schede già incontrato in questo capitolo.

Figura 8.8
Il *wizard Sundae utilizza il modello del wizard generico per guidare gli utenti nella creazione di una coppa di gelato.*

La proprietà .Tag

È possibile utilizzare la proprietà .Tag per memorizzare le informazioni che riguardano i controlli necessari per un programma. Tutti i controlli prevedono una proprietà .Tag, che non è utilizzata da Visual Basic ma deve essere utilizzata per identificare gli oggetti nei programmi. Il tipo di dati della proprietà di un controllo è la stringa e il suo valore predefinito è la stringa vuota ("").

Per creare il wizard Sundae ho cominciato con i file del wizard generico, e ho seguito i passi da 1 a 4 per la creazione di un wizard basato su quello generico. A quel punto la procedura SetUpWiz era la seguente:

```
Private Sub SetUpWiz()
   'Pannello 1
   ItemsInPanel 4
   'Pannello 2
   ItemsInPanel 4
   'Pannello 3
   ItemsInPanel 4
End Sub
```

In altre parole il wizard Sundae prevede tre pannelli, ciascuno dei quali contiene quattro controlli. Durante la pianificazione del wizard mi sono anche accorto che volevo un controllo aggiuntivo in comune a tutti i pannelli: il controllo `Image` che visualizza un bitmap della coppa gelato (la si può vedere in tutta la sua bellezza nella Figura 8.8).
Per modificare la codifica del wizard, in modo da includere un altro controllo in tutti i pannelli, ho cambiato la costante relativa al livello dei moduli in SillyWiz.Bas da:

```
Const DesignTimeControls = 4 'numero dei controlli su tutti i Wizard
```

a

```
Const DesignTimeControls = 5 'numero dei controlli su tutti i Wizard
```

Dato che ho aggiunto un controllo comune a tutti i pannelli il wizard Sundae prevede ora tre pulsanti, una riga e il controllo d'immagine; la proprietà `.Tag` relativa ai controlli (illustrata nel punto 5) deve iniziare da 6 (e non da 5). Oltre ai controlli comuni, il wizard Sundae prevede un riquadro e una matrice di pulsanti opzione con tre pulsanti su ciascun pannello, per un totale quindi di quattro controlli per pannello.
Il wizard Sundae a questo punto è in grado di funzionare e visualizza i suoi pannelli come desiderato. Per dimostrare che si possono catturare i valori relativi ai controlli del wizard ho aggiunto del codice alla routine `ShutDown`, mediante la quale si visualizza una finestra di messaggio che mostra i valori selezionati dall'utente (si vedano il Listato 8.3 e la Figura 8.8).

Listato 8.3 *Cattura dei valori dai controlli del wizard.*

```
Private Sub ShutDown()
   Dim X As Integer, Msg As String, Container As String,
      IceCream As String, Toppings As String
   For X = 0 To 2
    With ThisWizard
      If .optContainer(X) Then Container = .optContainer(X).Caption
      If .optlceCream(X) Then IceCream = .optlceCream(X).Caption
      If .optToppings(X) Then Toppings = .optToppings(X).Caption
    End With
   Next X
   If Container = "" And IceCream = "" And Toppings = "" Then
      Msg = "Non hai scelto nulla!"
   Else
      Msg = "La tua coppa è così composta:" + vbCrLf + vbCrLf
```

```
      Msg = Msg + "Servito in:  " + Container + vbCrLf
      Msg = Msg + "Gusto di gelato:  " + IceCream + vbCrLf
      Msg = Msg + "Con aggiunta di:  " + Toppings
   End If
   MsgBox Msg, vbInformation, "Buon appetito!"
   Unload ThisWizard
   End
End Sub
```

(Sul CD trovate l'originale inglese. Qui abbiamo tradotto le stringhe per intendere meglio il significato e per chi volesse personalizzare il wizard.) Il wizard ora funziona perfettamente, anche se sfortunatamente non è capace di servire una vera coppa gelato! Purtroppo non siamo ancora su Star Trek. La prima volta che ho provato il wizard Sundae ho notato una cosa che non mi piaceva. Quando si faceva clic su *Finish* il wizard scompariva prima che venisse visualizzato il messaggio con il contenuto della coppa gelato. Ho deciso che è meglio lasciare il pannello finale del wizard sullo schermo mentre si visualizza il messaggio (si veda la Figura 8.8).
È stato facile effettuare questa modifica; ho semplicemente fatto diventare un commento la riga che richiama i metodi .Hide da ChangePanel, posta appena prima della procedura ShutDown; ChangePanel è la routine che permette all'utente di spostarsi tra i pannelli del wizard. Proseguiamo ora nella lettura con una spiegazione della logica del wizard!

Analisi del codice del wizard

Vediamo le dichiarazioni a livello di modulo relative a Wizard.Bas:

```
Option Explicit
Dim LastPanel As Integer, Panelltems() As Integer,
   ThisPanel As Integer, ThisWizard As New frmWizard
Const DesignTimeControls = 4 'numero dei controlli su tutti i wizard
```

ThisPanel e LastPanel sono variabili che rappresentano rispettivamente il numero del pannello corrente e dell'ultimo pannello nel wizard. Panelltems( ) è un array dinamico di interi che viene utilizzato per tenere traccia dei controlli sul pannello corrente, con un artificio: PanelItems(x), dove *x è* il pannello corrente, è definito dal numero di controlli del pannello corrente più il valore di Panelltems(x-1). Per esempio, dato che Sundae prevede tre pannelli, ciascuno con quattro voci, si ha PanelItems(3) = 12.
DesignTimeControls identifica il numero di controlli che sono comuni a tutti i pannelli del wizard; vale uno meno del valore del primo per quei controlli la cui proprietà .Visible viene elaborata dal wizard.
La dichiarazione di ThisWizard come New frmWizard significa che ThisWizard è un'istanza di frmWizard; questo è un aspetto imFortante da comprendere. Tutta la codifica relativa al wizard si trova nel modulo .Bas e si applica all'istanza ThisWizard, non a frmWizard. (La codifica di frmWizard è costituita da una singola chiamata di riga al modulo della codifica relativa a quando si fa clic su *Back, Next* oppure *Finish).*

Il progetto del wizard generico ha inizio da Sub Main:

```
Public Sub Main()
   Dim I As Integer
   LastPanel = 0
   SetUpWiz
   'Mostra il primo pannello
   ThisPanel = 1
   ThisWizard.cmdBack.Enabled = False
   'Popola il form
   For I = 0 To ThisWizard.Controls.Count - 1
      If Val(ThisWizard.Controls(I).Tag) > DesignTimeControls And
         Val(ThisWizard.Controls(I).Tag) <= DesignTimeControls + _
         PanelItems(1) Then
            ThisWizard.Controls(I).Visible = True
      End If
   Next I
   ThisWizard.Show
End Sub
```

Sub Main richiama in primo luogo la procedura SetUpWiz, che consente la definizione del numero di pannelli e di voci per pannello. Nel caso del wizard generico si possono trovare tre pannelli e nessun controllo:

```
Private Sub SetUpWiz()
   'Pannello 1
   ItemsInPanel 0
   'Pannello 2
   ItemsInPanel 0
   'Pannello 3
   ItemsInPanel 0
End Sub
```

La procedura ItemsInPanel che viene richiamata da SetUpWiz legge i valori nell'array dinamico ItemsInPanel, come descritto in precedenza:

```
Private Sub ItemsInPanel(HowMany As Integer)
   LastPanel = LastPanel + 1
   ReDim Preserve PanelItems(1 To LastPanel)
   If LastPanel > 1 Then
      PanelItems(LastPanel) = HowMany + PanelItems(LastPanel - 1)
   Else
      PanelItems(LastPanel) = HowMany 'solo per il primo pannello
   End If
End Sub
```

Una volta ritornati a Sub Main, il resto della procedura predispone l'aspetto del primo pannello. Definisce come False la proprietà .Enabled di cmdBack dato che non si può procedere all'indietro rispetto al primo pannello; utilizza le proprietà .Tag di tutti i controlli nell'insieme dei controlli in ThisWizard per definire come True la proprietà .Visible di tutti i controlli relativi al primo pannello:

```
For I = 0 TO ThisWizard.Controls.Count - 1
   If Val(ThisWizard.Controls(I).Tag) > DesignTimeControls And _
```

```
        Val(ThisWizard.Controls(I).Tag) <= DesignTimeControls + _
        PanelItems(I) Then
            ThisWizard.Controls(I).Visible = True
        End If
Next I
```

Infine, il wizard è visualizzato mediante il suo metodo .Show.
Viene richiamata la procedura ChangePanel ogni volta che l'utente preme cmdBack (argomento uguale a 1) oppure cmdNext (argomento 2) in frmWizard (è già stato detto che queste sono le uniche righe di codice presenti in frmWizard):

```
Private Sub cmdNext_Click()
    ChangePanel 1
End Sub

Private Sub cmdBack_Click()
    ChangePanel 2
End Sub
```

Il Listato 8.4 mostra la routine ChangePanel.

Listato 8.4 *Modifica dei pannelli,*

```
Public Sub ChangePanel(Which As Integer)
    Select Case Which
        Case 1                      'Next o Finish
            If ThisPanel = 1 Then
                ThisWizard.cmdBack.Enabled = True
            End If
            ThisPanel = ThisPanel + 1
            If ThisPanel = LastPanel Then
                ThisWizard.cmdNext.Caption = "&Finish"
            ElseIf ThisPanel > LastPanel Then
                ThisWizard.Hide
                ShutDown
            End If
            SetNewPanel ThisPanel - 1, NextPanel(ThisPanel - 1)
        Case 2 'Back
            If ThisPanel = LastPanel Then
                ThisWizard.cmdNext.Caption = "&Next"
            End If
            SetNewPanel ThisPanel, PrevPanel(ThisPanel)
            ThisPanel = ThisPanel - 1
            If ThisPanel = 1 Then
                ThisWizard.cmdBack.Enabled = False
            End If
        Case Else                   'non si deve arrivare qui
            MsgBox "Errore interno del wizard!"
    End Select
End Sub
```

La procedura si occupa di abilitare il pulsante *Back* se il wizard non sta visualizzando il primo pannello, di modificare la didascalia *Next* in *Finish* sull'ultimo pan-

nello e di richiamare la procedura ShutDown quando si fa clic su *Finish*. Si occupa anche di incrementare (oppure di decrementare) il contenuto di ThisPanel, se necessario.
SetNewPanel svolge il lavoro vero e proprio di gestire la visibilità del nuovo pannello. Viene richiamato a prescindere dalla direzione nella quale ci si sta muovendo nel wizard, anche se con diversi argomenti a seconda che si vada in avanti o all'indietro. Le funzioni NextPanel e PrevPanel, a loro volta richiamate dalla chiamata della procedura SetNewPanel, aumentano oppure diminuiscono l'argomento relativo al prossimo pannello oppure a quello precedente, in un modo che consente di escludere un errore di fuori intervallo (per esempio, di andare indietro rispetto al pannello zero oppure di passare al pannello quattro in un wizard di tre pannelli):

```
Private Function NextPanel(01dPanel As Integer) As Integer
    If OldPanel = LastPanel Then
        NextPanel = OldPanel
    Else
        NextPanel = OldPanel + 1
    End If
End Function

Private Function PrevPanel(01dPanel As Integer) As Integer
    If OldPanel = 1 Then
        PrevPanel = OldPanel
    Else
        PrevPanel = OldPanel - 1
    End If
End Function
```

SetNewPanel è mostrata nel Listato 8.5.

Listato 8.5 *Definizione di un nuovo pannello.*

```
Private Sub SetNewPanel(oldP As Integer, newP As Integer)
  Dim I As Integer
  If newP > oldP Then                  'Avanti
    For I = 0 To ThisWizard.Controls.Count - 1
      If Val(ThisWizard.Controls(I).Tag) > PanelItems(oldP) _
        + DesignTimeControls And Val(ThisWizard.Controls(I).Tag)_
        <= PanelItems(newP) + DesignTimeControls Then
          ThisWizard.Controls(I).Visible = True
      Else
        If Val(ThisWizard.Controls(I).Tag) > DesignTimeControls Then
          ThisWizard.Controls(I).Visible = False
        End If
      End If
    Next I
  ElseIf newP > 1 Then           'Indietro
    For I = 0 To ThisWizard.Controls.Count - 1
      If Val(ThisWizard.Controls(I).Tag) <= PanelItems(newP) _
        + DesignTimeControls And Val(ThisWizard.Controls(I).Tag)_
        > PanelItems(PrevPanel(newP)) + DesignTimeControls Then
          ThisWizard.Controls(I).Visible = True
      Else
```

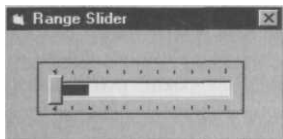
```
        If Val(ThisWizard.Controls(I).Tag) > DesignTimeControls Then
          ThisWizard.Controls(I).Visible = False
        End If
      End If
    Next I
    Else 'caso speciale indietro al primo pannello
    For I = 0 To ThisWizard.Controls.Count - 1
      If Val(ThisWizard.Controls(I).Tag) <= PanelItems(newP) _
        + DesignTimeControls And Val(ThisWizard.Controls(I).Tag) _
        > DesignTimeControls Then
          ThisWizard.Controls(I).Visible = True
      Else
        If Val(ThisWizard.Controls(I).Tag) > DesignTimeControls Then
          ThisWizard.Controls(I).Visible = False
        End If
      End If
    Next I
  End If
End Sub
```

Come nel caso del codice di Sub Main che definisce il primo pannello, questo codice utilizza l'insieme di controlli di ThisWizard per stabilire la visibilità dei controlli, in funzione della direzione del movimento, di un nuovo pannello e della proprietà .Tag di ciascun controllo. Anche se la logica di tutto questo può apparire complessa, diventa più facile da comprendere se si lavora con qualche esempio. Una volta visti gli esempi, sarete in grado di comprendere la logica dei wizard e sarete quindi pronti per creare i vostri wizard personali!

Utilizzo delle demo ProgressBar e Slider

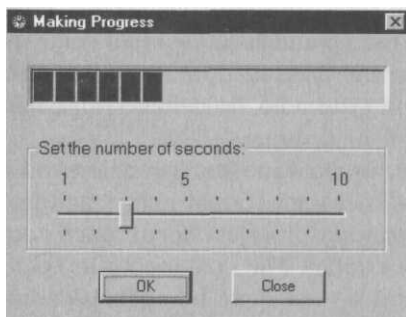
È abbastanza chiaro stabilire quali proprietà bisogna definire per far funzionare i controlli ProgressBar *e* Slider. *In questa applicazione campione, disponibile sul CD-ROM allegato al libro come ProgSlid. Vbp,* Slider *è utilizzato per definire il numero di secondi per i quali deve funzionare* ProgressBar. *Slider è equipaggiato con un cursore che l'utente fa scivolare per selezionare un valore. Le proprietà delle quali ci si deve occupare in questa piccola applicazione sono* Max *e* Min, *che definiscono i valori superiore e inferiore di* Slider, *e* Value, *che definisce il valore fissato dall'utente quando muove il cursore di* Slider.

Grazie alle proprietà del controllo Slider è anche possibile definire un intervallo per i valori che l'utente può selezionare. Per configurare Slider in questo modo ci si deve assicurare che la proprietà SelectRange sia fissata a True, quindi si definiscono le proprietà SelLength e SelStart che riguardano rispettivamente la lunghezza della selezione di valori e il valore di partenza della selezione. Per esempio, la Figura 8.9 mostra *Slider con* le seguenti impostazioni: SelectRange = True, SelLength = 2 e SelStart = 0. Come si può notare dalla figura, l'intervallo della selezione è di due tacche e il valore predefinito di Slider quando l'applicazione viene lanciata è compreso tra O e 2.

Figura 8.9 *Uno Slider per selezionare un valore in un intervallo.*

Il controllo ProgressBar visualizza "piastrine" azzurre che segnalano graficamente il passaggio del tempo. L'applicazione demo si affida alle proprietà Min e Max per definire i valori minimo e massimo dell'intervallo, oltre a una variabile Value che risulta disponibile solo in runtime. Una volta avviato un nuovo progetto, ho inser1to in un modulo uno Slider, un ProgressBar e due pulsanti di comando, *OK e Close.*
L'azione principale di questo progetto viene svolta dopo che l'utente ha utilizzato *Slider per* selezionare il numero di secondi che deve impiegare ProgressBar per completarsi ed ha fatto clic su *OK.* L'evento clic sul pulsante *OK* stabilisce l'istante in cui si definisce la durata di ProgressBar e si attiva Timer1. L'evento Timer di Timer"! viene attivato per il numero di volte stabilito da cmdOK_Click, sulla base della proprietà Timer1 . Interval e della variabile Time (che dichiara il tempo totale di esecuzione di ProgressBar).

```
Private Sub cmdOK_Click()
    Time = Slider1.Value * 1000
    Timer1.Interval = 100
    ProgressBar1.Max = Time
    Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()
    TimeElapsed = TimeElapsed + Timer1.Interval
    If TimeElapsed > Time Then
        Timer1.Enabled = False
         ProgressBar1.Value = 0
         TimeElapsed = 0
        Exit Sub
    End If
    ProgressBar1.Value = TimeElapsed
End Sub
```

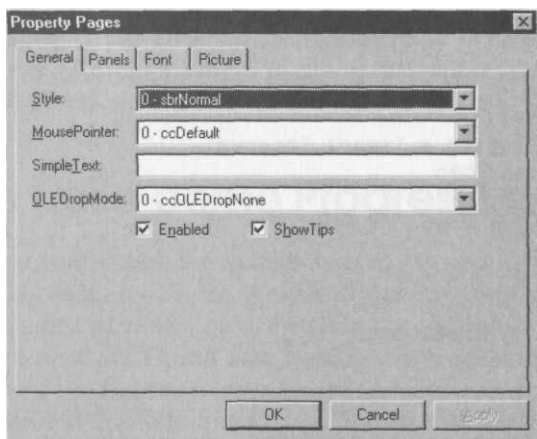
Quando si esegue l'applicazione, la finestra corrispondente è quella mostrata in Figura 8.10.

Figura 8.10 *Questa applicazione consente di osservare i secondi che trascorrono.*

Utilizzo della demo editor di testo

Questa applicazione utilizza il controllo RichTextBox come parte integrante di una semplice applicazione editor di testo. Non è molto ricca di funzioni, può solo aprire, memorizzare e stampare file di testo in formato .Rtf oltre a modificare gli attributi del font.
Non dovrebbe comunque essere difficile espandere l'applicazione, aggiungendo di volta in volta una nuova funzione per l'elaborazione di testi; il controllo RichText-Box lo consente. Le possibilità di manipolazione dei font sono piuttosto notevoli e consentono a questo controllo di dare parecchi punti alle tradizionali caselle di testomultiriga.

L'applicazione RichTextBox include anche i controlli ToolBar, ImageList *e* StatusBar; *l'applicazione si trova nel CD-ROM allegato al libro nella directory relativa al Capitolo 8, sotto il nome di RichText. Vbp.*

Una volta avviato un nuovo progetto, ho fatto doppio clic sul controllo StatusBar per aggiungerne uno al form. La barra di stato è per definizione allineata rispetto al fondo del modulo con un riquadro visibile (si veda la Figura 8.11). Per modificare le proprietà della barra di stato ho selezionato *Custom* nella finestra *Properties,* per accedere alla *Property Pages* relativa al controllo, come mostrato in Figura 8.11.

Figura 8.11 *// controllo StatusBar predefinito* e *la sua Property Pages.*

Nella *Property Pages* di *StatusBar* mi sono spostato sulla scheda *Panels,* dove ho definito le seguenti proprietà per il primo pannello: *Alignment* come *1-Center, Style* come *5-Time* e *AutoSize* come 1-*Spring.* A questo punto ho premuto il pulsante *InsertPanel* per aggiungere un secondo pannello, ho definito *Alignment* e *AutoSize* come nel primo pannello, mentre *Style* ha il valore *6-Date.*
Successivamente ho trascinato un controllo RichTextBox all'interno del modulo, l'ho tirato fino a fargli assumere le dimensioni corrette, ho definito come True la proprietà *DisableNoScroll,* ho modificato la proprietà ScrollBars con il valore 3-Both. Una volta definito un semplice menu *File* contenente le voci *Open, Save, Print* e *Exit,* oltre a un menu *Style* con la voce *Font,* ho aggiunto al modulo i con-

trolli ToolBar, ImageList e CommonDialog. Ho caricato quattro immagini bitmap per rappresentare *Operi, Save, Print* e *Foni* all'interno di ImageList, utilizzando la scheda *Images* della *Property Pages* del controllo, come mostrato in Figura 8.12.

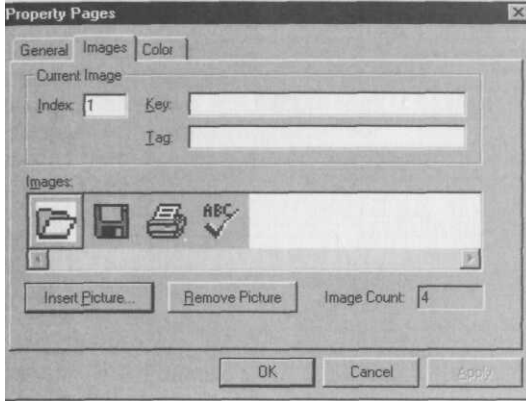
Figura 8.12
Ho utilizzato la scheda Images per caricare quattro immagini nel controllo ImageList.

Mi sono quindi spostato sulla pagina della scheda Colors *per definire la proprietà* BackColor *come grigio chiaro e la proprietà* MaskColor *come bianco (si veda la Figura 8.13). Se non si definiscono queste proprietà nel modo indicato, le immagini hanno un aspetto strano quando vengono caricate sui pulsanti della barra di stato: sembrano disabilitate.*

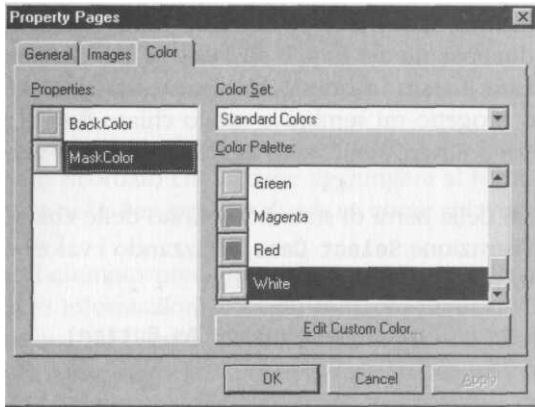
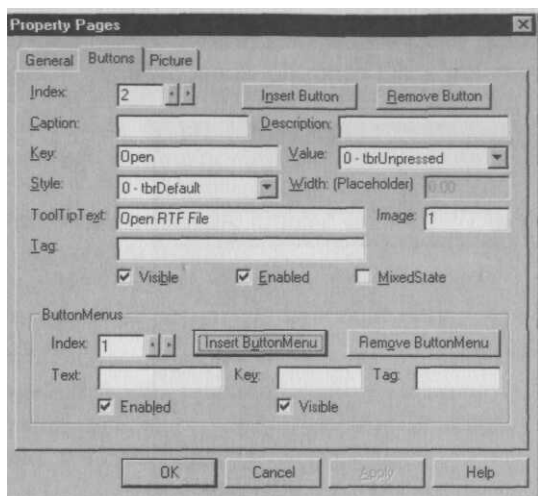
Figura 8.13
Si può vedere la pagina della scheda Colors che definisce BackColor e MaskColor delle immagini.

Per caricare le immagini sui pulsanti della barra di stato, ho aperto la finestra *Property Pages* relativa al controllo ToolBar (selezionando la proprietà *Custom* oppure con un clic destro sul controllo). Ho collegato il controllo ImageList al controllo ToolBar utilizzando la casella di riepilogo a discesa posta vicino alla proprietà *ImageList* nella, scheda *General.* Successivamente mi sono spostato sulla scheda *Buttons* per creare i pulsanti della barra di stato e assegnare le immagini corrispondenti (Figura 8.14).

Figura 8.14 *Si utilizza la scheda Buttons per inserire i pulsanti e per assegnare a questi delle immagini.*

Ho premuto il pulsante *Inserì Button* per creare il primo pulsante, quindi ho inserito *Open* nella casella di testo *Key* e *Open RTF File* nella casella *ToolTipText;* ho stabilito la prima immagine del controllo *ImageList* relativa al pulsante scrivendo 1 nella casella *Image.* Ho premuto di nuovo *Insert Button* per aggiungere il pulsante successivo, assegnando a questo le stesse proprietà del primo. Ho continuato fino a completare il caricamento di tutti e quattro i pulsanti.

Quando si esegue questa applicazione editor di testo, l'utente può scrivere quello che vuole all'interno di RichTextBox, può selezionare parti di testo per le quali modificare font, colore e dimensione del font, e così via. Inoltre, l'utente può aprire file di tipo .Rtf, memorizzare il testo in questo formato e stampare quello che ha scritto. Il codice di questo progetto mi sembrava molto chiaro fino al momento in cui ho cominciato a crearlo; poi però ho dovuto aggiungere alcuni accorgimenti di cui parleremo più avanti.

Per far funzionare i pulsanti della barra di stato ho definito delle chiamate alle voci del menu all'interno di un'istruzione Select Case, utilizzando i valori Key che sono definiti nel momento in cui vengono caricati i pulsanti:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As Button)
    Select Case Button.Key
        Case "Open"
            mnuOpen_Click
        Case "Save"
            mnuSave_Click
        Case "Print"
            mnuPrint_Click
        Case "Font"
            mnuFont_Click
    End Select
End Sub
```

Il compito successivo è stato quello di scrivere il codice che attiva ogni voce del menu. Il Listato 8.6 mostra la parte del codice che consente la modifica del font e dei suoi attributi:

Listato 8.6 *Modifica difont e attributi.*

```
private Sub mnuFont_Click()
   CommonDialog1.FontName = RichTextBox1.SelFontName
   CommonDialog1.FontSize = RichTextBox1.SelFontSize
   CommonDialog1.FontBold = RichTextBox1.SelBold
   CommonDialog1.Fontltalic = RichTextBox1.Selltalic
   CommonDialog1.Color = RichTextBox1.SelColor
   CommonDialog1.FontStrikethru = RichTextBox1.SelStrikethru
   CommonDialog1.FontUnderline = RichTextBox1.SelUnderline
   CommonDialog1.CancelError = True
   On Error GoTo ErrHandle
   CommonDialog1.Flags = cdlCFBoth + cdlCFWYSIWYG + cdlCFEffects
   CommonDialog1.ShowFont
   On Error GoTo 0
   RichTextBox1.SelFontName = CommonDialog1.FontName
   RichTextBox1.SelFontSize = CommonDialog1.FontSize
   RichTextBox1.SelBold = CommonDialog1.FontBold
   RichTextBox1.Selltalic = CommonDialog1.Fontltalic
   RichTextBox1.SelStrikethru = CommonDialog1.FontStrikethru
   RichTextBox1.SelUnderline = CommonDialog1.FontUnderline
   RichTextBox1.SelColor = CommonDialog1.Color
   Exit Sub
ErrHandle:
   If Not Err = cdlCancel Then Resume Next
End Sub
```

Si utilizza il dialogo comune dei font di Windows per elaborare gli input che riguardano i font. (Ho ricordato che si deve aggiungere al forni un controllo dei dialoghi comuni, oppure no?). Per saperne di più su come si lavora con i dialoghi comuni si veda il Capitolo 7.
Nel codice dell'esempio precedente, il controllo dei dialoghi comuni viene prima imbottito con le informazioni correnti di RichTextBox. A patto che l'utente faccia una selezione qualsiasi senza cancellare tutto, le impostazioni nel dialogo dei font sono assegnate come valori delle corrispondenti proprietà di *RichTextBox.* A questo proposito, se non viene selezionato del testo prima che sia aperto il dialogo comune, le informazioni sui font risultano modificate a partire dal punto di inserimento. Il Listato 8.7 mostra come stampare il contenuto del controllo:

Listato 8.7 *Stampa del contenuto di un RichTextBox.*

```
Private Sub mnuPrint_Click()
   CommonDialog1.CancelError = True
   On Error GoTo ErrHandle
   CommonDialog1.Flags = cdlPDNoPageNums
   If RichTextBox1.SelLength = 0 Then
      CommonDialog1.Flags = CommonDialog1.Flags + cdlPDAllPages
```

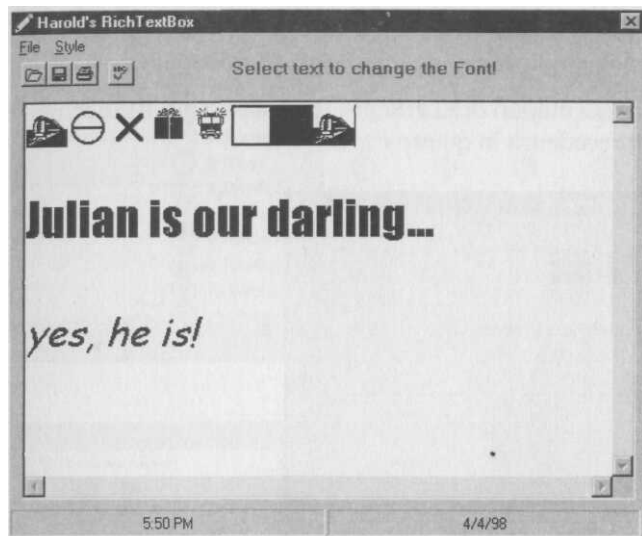
```
   Else
      CommonDialog1.Flags = CommonDialog1.Flags + cdlPDSelection
   End If
   CommonDialog1.ShowPrinter
   Printer.Print
   RichTextBoxI.SelPrint Printer.hDC 'Contesto di dispositivo stampante
   Printer.EndDoc
Exit Sub
ErrHandle:
  If Not Err = cdlCancel Then Resume Next
End Sub
```

Questo codice *utilizza* il dialogo comune per definire l'oggetto stampante corrente, quindi inizializza l'oggetto Printer utilizzando il suo metodo .Print. Il metodo .SelPrint di RichTextBox viene chiamato con il contesto di dispositivo di Printer come argomento (non il contesto di dispositivo restituito dal dialogo comune nella codifica di esempio). Infine viene chiamato il metodo .EndDoc della stampante per terminare la stampa. Il Listato 8.8 mostra il codice relativo alle voci *Open* e *Save*

Listato 8.8 *Codifica relativo ad apertura e memorizzazione di un RichTextBox.*

```
Private Sub mnuOpen_Click()
   CommonDialog1.CancelError = True
   On Error GoTo ErrHandle
   CommonDialog1.Filter =
      "Rich Text File (*.Rtf)|*.rtf|A11 Files (*.*)|*•*"
   CommonDialog1.Flags = cdlOFNPathMustExist + cdlOFNHideReadOnly _
      +cdlOFNFileMustExist
   CommonDialog1.ShowOpen
   RichTextBox1.LoadFile(CommonDialog1.filename)
   On Error GoTo 0
   Exit Sub
ErrHandle:
   If Not Err = cdlCancel Then
      Resume Next
   End If
End Sub

Private Sub mnuSave_Click()
   CommonDialog1.CancelError = True
   On Error GoTo ErrHandle
   CommonDialog1.Filter =
      "Rich Text File (*.Rtf)|*.rtf|A11 Files (*.*)|*.*"
   CommonDialog1.Flags =
      cdlOFNCreatePrompt + cdlOFNOverwritePrompt + _
      CdlOFNPathMustExist + cdlOFNHideReadOnly
   CommonDialog1.ShowSave
   RichTextBox1.SaveFile  (CommonDialog1.filename)
   On Error GoTo 0
   Exit Sub
ErrHandle:
   If Not Err = cdlCancel Then Resume Next
End Sub
```

Se si considera la potenzialità dei suoi metodi predefiniti, il controllo *RichTextBox* può essere considerato straordinariamente potente! La Figura 8.15 mostra qualcosa di quello che si può ottenere.

Figura 8.15 *RichTextBox sfoggia quello che può fare.*

CoolBar

La cosiddetta CoolBar *(il nome non richiede alcun commento, se non per dire che è veramente favoloso) è un controllo contenitore di strumenti che consente di personalizzare la disposizione di diversi gruppi di controlli. Grazie a questo controllo è possibile creare barre strumenti personalizzate, simili a quella di Internet Explorer.*

Un controllo CoolBar *è un contenitore costituito da una o più "band"; ogni band può essere dimensionata e sistemata a piacere dall'utente. Sotto ceni aspetti è simile al controllo* ToolBar; *analogamente a questo lavora in genere con un controllo* ImageList *associato, come illustrato nel paragrafo precedente.*

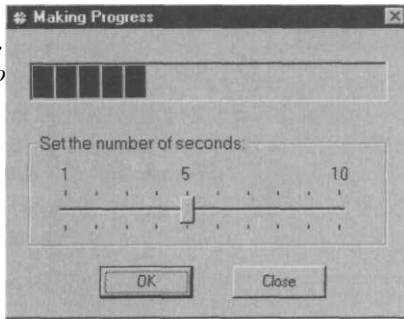
FlatScrollBar

FlatScrollBar *è una novità di VB6 e lavora in modo simile a una convenzionale barra di scorrimento (o* Slider) *ma la sua interfaccia è stata migliorata. Questo controllo può avere l'aspetto di una barra di scorrimento standard di tipo tridimensionale, quello di una barra a due dimensioni oppure essere di tipo piatto con le frecce che diventano tridimensionali quando il mouse si trova a passarvi sopra.*

FlatScrollBar può essere utilizzato per:

- Mettere a disposizione più stili di interfaccia mentre si utilizza un singolo controllo.
- Imitare l'aspetto e le funzionalità di Internet Explorer.
- Aggiungere alle applicazioni un'interfaccia in stile multimediale.

La Figura 8.16 mostra l'utilizzo di FlatScrollBar al posto di Slider nella demo ProgSlid introdotta in precedenza in questo capitolo.

Figura 8.16 *Sipuòutilizzare ilnuovocontrollo FlatScrollBar alpostodiSlider oppure di una convenzionale barra discorrimento.*

Visualizzazione delle gerarchie: i controlli ListView e TreeView

Il controllo ListView viene utilizzato per visualizzare informazioni di tipo gerarchico; questo controllo è utilizzato spesso in combinazione con TreeView e consente di mostrare elenchi di dati. I due controlli dovrebbero risultare familiari agli utenti di Windows, in quanto l'interfaccia di Explorer (Gestione risorse o Esplora risorse) è costituita essenzialmente da un controllo TreeView sulla sinistra e da un controllo *ListView* sulla destra.

Ilprogetto campione, presente sul CD-ROM come TLView.Vbp, utilizza un controllo TreeView *nella pane sinistra del suo form e* ListView *nella parte destra, proprio come Esplora risorse. Inoltre ho inserito due controlli* ImageList *e una matrice dipulsanti di opzione a quattro posizioni che consentono all'utente di selezionare in runtime lepossibili modalità previste da* ListView *(si vedano lefigure 8.17 e 8.18).*

L'applicazione dimostrativa visualizza un intervallo ordinato di anni. Se si espande uno degli anni compaiono i mesi, poi ci sono i giorni del mese. Se non altro, il programma può servire per scoprire in che giorno della settimana cade il vostro compleanno nel 2005, oppure per sapere se nel 2004 è previsto il 29 febbraio! Il controllo ListView è riempito con i giorni del mese che viene espanso.

Figura 8.17

Quando viene espanso, il controllo TreeView fornisce un modo eccellente per visualizzare informazioni ordinate.

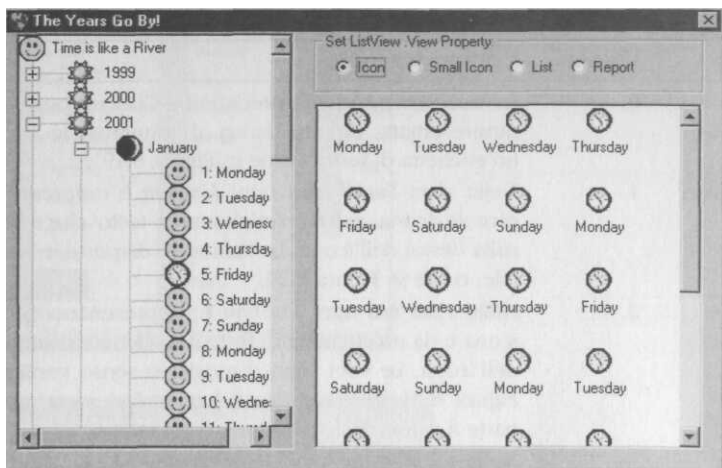



Figura 8.18

Questa è l'applicazione completa che utilizza i controlli TreeView e ListView.

Tabelld 8.3 *Impostazioni. View di ListView.*

Costante	Valore	Comportamento
IvwIcon	0	La vista *Icon è* quella predefinita. Ciascun oggetto *ListItem è* rappresentato da una icona di dimensione standard e da un'etichetta di testo, come in Figura 8.19.
IvwSmallIcon	1	Nella vista *Small Icon* ogni *ListItem* e rappresentato da una piccola icona e da un'etichetta di testo che viene mostrata sulla destra dell'icona. Le voci sono disposte in senso orizzontale, come in Figura 8.20.
IvwList	2	Nella vista *List* ogni *ListItem* e rappresentato da una piccola icona e da un'etichetta di testo che viene mostrata sulla destra dell'icona. Le voci sono disposte in senso verticale, come in Figura 8.21. Le viste *Small Icon* e *List* sono molto simili, a parte il senso di disposizione delle icone.
IvwReport	3	Nella vista *Report* ogni *ListItem* viene visualizzato con la sua piccola icona e l'etichetta di testo. E possibile aggiungere altre informazioni *(SubItems)* a ciascuna voce. Le icone, le etichette di testo e le altre informazioni vengono incolonnate; la prima colonna a sinistra contiene le piccole icone ed e seguita dalle etichette di testo. La Figura 8.22 mostra un controllo *ListView* definito come vista *Report.*

Figura 8.19
Quando il controllo ListViewèdefinito in vista Icon, gli oggetti sono rappresentati icone di grandi dimensioni.

Figura 8.20
Nella vista Small Icon gli oggetti sono rappresentati da piccole icone disposte in senso orizzontale.

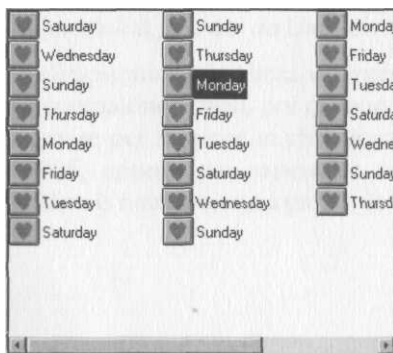
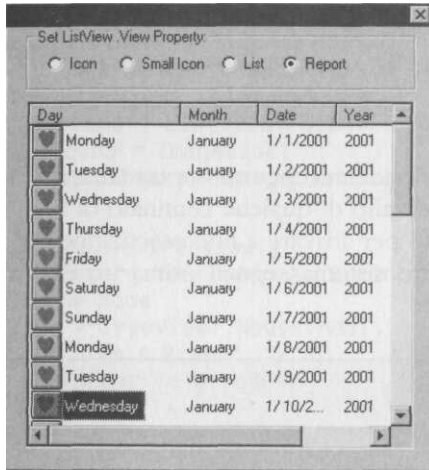



Figura 8.21

Nella vista List di oggetti sono rappresentati da piccole icone disposte in senso verticale.

Figura 8.22

Le intestazioni di colonna e le voci SubItems sono visualizzate solo quando un controllo ListView è definito come vista Report.

Se si utilizza la proprietà .View di ListView, i suoi controlli possono essere commutati tra le quattro diverse condizioni possibili. La Tabella 8.3 mostra i valori consentiti da *.View* e descrive l'aspetto corrispondente previsto dal controllo ListView.

Un aspetto imFortante da considerare sulla proprietà *.View* riguarda il fatto che solo quando un ListView si trova in modalità *Report* può visualizzare le intestazioni delle colonne e le voci SubItems. Torneremo sull'argomento, ma dovreste avere già capito dalla Tabella 8.3 che ListView contiene un insieme di `ListItems`.

Ho inserito due controlli ImageList nel progetto, da utilizzare come "librerie" per i controlli TreeView e ListView. Ho avuto bisogno di due `ImageList` perché ho voluto che le immagini che rappresentano i ListItems di ListView fossero più piccole di quelle utilizzate nei nodi di `TreeView`. (Le immagini memorizzate in un controllo ImageList devono infatti avere tutte le stesse dimensioni.) Avete già visto come utilizzare i controlli ImageList in questo stesso capitolo.

Nonostante il fatto che sia possibile definire molte proprietà di TreeView *e di* ListView *nella finestra* Properties *oppure mediante il codice, devo dire che per questi controlli preferisco avere il più possibile a che fare con la codifica. L'unica proprietà definita nella finestra* Properties *è stata* TreeView1.Style, *che ho impostato come "Treelines, Plus/Minus, Picture, and Text", utilizzando un menu a discesa. Questa impostazione significa che* TreeView *visualizza una serie di ramificazioni, i simboli più e meno per indicare se un nodo è espanso oppure compresso, un'immagine (se ne è stata assegnata una al nodo) e del testo. La costante e il valore equivalenti, se si vuole definire questa proprietà nel codice,* sono tvwTreelinesPlusMinusPictureText *e 7.*

A questo proposito il Capitolo 10 contiene un altro esempio che utilizza il controllo TreeView, in questo caso per visualizzare parti del Registro di configurazione.
Per cominciare, ho creato le costanti a livello di forni che stabiliscono l'intervallo di anni che l'applicazione deve visualizzare:

```
Option Explicit
Const StartYear = 1999
Const StopYear = 2003
```

È possibile definire i valori che si desiderano per queste costanti, anche se occorre ricordare che se si imposta un intervallo di qualche centinaio di anni si dovrà poi aspettare fino al prossimo millennio per arrivare a fine esecuzione del programma. (In realtà, è probabile che il vostro sistema segnali prima un esaurimento della memoria).

La funzione Format

Ancora un preliminare prima di iniziare. Nel codice che segue ho utilizzato più volte la funzione Format; si possono notare molti modi diversi di applicare espressioni relative alle date, definite dall'utente, nelle chiamate della funzione Format.
In effetti Format è uno dei grandi piccoli segreti di Visual Basic. Questo potente "mulo da soma" è talmente modesto che non ci si rende conto di quello che può fare fino a quando non si prova a farci un giro sopra. Format è in grado di prendere una grande varietà di input che rigurgita in funzione di espressioni di formattazione già disponibili o che possono essere definite dall'utente. L'aspetto più intrigante è legato alle espressioni di formattazione che possono essere definite dall'utente. Per avere maggiori informazioni si vedano gli argomenti User-Defined Date/Time Formats, User-Defined Numeric Formats e User-Defined String Formats nella guida in linea di VB.

Il Listato 8.9 mostra l'evento di caricamento del form e la routine che richiama, PopulateTreeView. Queste procedure collegano i controlli TreeView e ListView con le rispettive librerie ImageList, inseriscono un oggetto ColumnHeader in ListView1, un nodo principale in TreeView1 e nodi su cinque livelli in TreeView per ciascun anno incluso. (Nel codice si può notare che PopulateTreeView richiama un'altra procedura, AddMonths, per ciascun anno che viene creato).

Listato 8.9 *Stipare un TreeView.*

```
private Sub Form_Load()
   ' crea una variabile oggetto per l'oggetto ColumnHeader.
   Dim clmX As ColumnHeader
   TreeView1.ImageList = ImageList1
   PopulateTreeView

   ' Aggiungere ColumnHeaders. La larghezza delle colonne è la larghezza
   ' del controllo divisa per il numero di oggetti.
   Set clmX = ListView1.ColumnHeaders.
      Add(, , "Day", ListView1.Width / 5)
   Set clmX = ListView1.ColumnHeaders. _
      Add(, , "Month", ListView1.Width / 5)
   Set clmX = ListView1.ColumnHeaders. _
      Add(, , "Date", ListView1.Width / 5)
   Set clmX = ListView1.ColumnHeaders. _
      Add(, , "Year", ListView1.Width / 5)
   Set clmX = ListView1.ColumnHeaders. _
      Add(, , "Memo", ListView1.Width / 5)
   ListView1.Icons = ImageList1
   ListView1.SmallIcons = ImageList2
End Sub

Private Sub PopulateTreeView()
   Dim X As Integer, YearToAdd As String, NodX As Node, _
      TopNode As Node
   Set TopNode = TreeView1.Nodes.Add(, , , _
      "Time is like a River", 5, 6)
   For X = StartYear To StopYear
      YearToAdd = Str(X)
      Set NodX =TreeView1.Nodes.Add(TopNode,tvwChild, , _
         YearToAdd, 1,2)
      AddMonths NodX, X
   Next X
End Sub
```

È importante ricordare che il metodo .Add *agisce sull'oggetto* Nodes *di* TreeView *e non direttamente sul controllo* TreeView.

Tutti gli argomenti del metodo Nodes.Add sono opzionali, anche se di solito si include il quarto argomento, che stabilisce il testo collegato al nodo. Il primo e il secondo argomento rappresentano il "genitore" del nuovo nodo e il grado di parentela con questo genitore. Gli ultimi due argomenti sono i numeri indice delle immagini del nodo nel corrispondente controllo ImageList (quella normale e quando selezionata).

AddMonths è passato al nodo corrente e viene aggiunto l'anno su cui si stanno contando i mesi. L'anno è utilizzato più avanti per stabilire se tutte le date possibili sono da considerare valide; per esempio il 29 febbraio 2007 non è una data valida perché il 2007 non è bisestile.

```
Private Sub AddMonths(NodX As Node, WhichYear As Integer)
   Dim X As Integer, MonthToAdd As String,
```

```
      MonthNode As Node
   For X = 1 To 12
      MonthToAdd = ConvertMonth(X)
      Set MonthNode = TreeView1.Nodes.Add(NodX, _
         tvwChild, Str(X) & "/01/" & Str(WhichYear), _
         MonthToAdd, 3, 4)
      AddDays MonthNode, WhichYear, X
   Next X
End Sub
```

AddMonths avvia un ciclo che consente di aggiungere un nodo per ciascun mese. All'interno del ciclo la conversione del contatore di interi nella stringa del mese è eseguita in un modo piuttosto simpatico:

```
Private Function ConvertMonth(X As Integer) As String
   ConvertMonth = Format(Str(X) + "/01/1997", "mmmm")
End Function
```

Dato che i mesi sono gli stessi per ogni anno, si poteva aggiungere il numero corrispondente al mese in una arbitraria stringa data e utilizzare Format *con "mmmm" come espressione che ritornasse solo la rappresentazione in strìnga del mese.*

Viene utilizzato un terzo argomento nel metodo Nodes.Add, omesso nelle precedenti chiamate; si tratta dell'argomento key, una stringa univoca che può essere utilizzata più avanti per identificare il nodo. Questo argomento è in formato data standard anglosassone e prevede il valore del mese che si sta aggiungendo, il primo giorno del mese e l'anno. Per esempio, "3/01/99" indica il nodo del mese di marzo 1999. Si vedrà tra breve il motivo di questo parametro addizionale.
Ogni volta che viene creato il nodo di un mese, viene chiamata la procedura AddDays con gli argomenti nodo del mese, anno (passato dalle procedure di più alto livello) e mese. Vediamo AddDays:

```
Private Sub AddDays(MonthNode As Node, WhichYear As Integer, _
   WhichMonth As Integer)
   Dim X As Integer, ThisDay As String, DateStr As String
   X = 1
   Do While X <= 31
      DateStr = Str(WhichMonth) + "/" + Str(X) + "/" + _
         Str(WhichYear)
      If Not IsDate(DateStr) Then
           X = X + 1
      Else
         ThisDay = Format(DateStr, "dddd")
         ThisDay = Str(X) + ": " + ThisDay
         TreeView1.Nodes.Add MonthNode, tvwChild, , ThisDay, 5, 7
         X = X + 1
      End If
   Loop
End Sub
```

Questa procedura cicla su tutti i possibili giorni del mese. Per ciascun giorno crea la stringa della data corrente (per esempio, 3/20/97 oppure 1/9/98). La funzione IsDate di Visual Basic consente di controllare se si tratta di una data valida (per esempio, 2/30 non è valida per nessun anno). Se la data viene confermata, si aggiunge un nodo corrispondente.
A questo punto, se si esegue il programma, si ottiene un piacevole e accurate albero di date, espandibile e ripiegabile su se stesso (si veda la Figura 8.16). Il passo successivo consiste nell'aggiungere il codice necessario per riempire ListView; la posizione consueta per fare questo si trova in corrispondenza dell'evento Expand di TreeView (si veda il Listato 8.10). Il programma dimostrativo è stato progettato in modo da riempire ListView1 solo quando si espande un nodo Month. Il codice che svolge questa operazione si basa sul fatto che ho aggiunto un parametro Key nel caso di nodi Month; se il valore della proprietà .Key del nodo passato alla procedura dell'evento Expand è costituito da una stringa vuota, il codice semplicemente esce dalla procedura. In caso contrario si tratta di un nodo Month, per cui è possibile riempire ListView con Listltems e Subltems. (Le voci Subltems sono visualizzate solo se ListView1.View è definito come IvwReFort.)

Listato 8.10 *Espansione di ListView.*

```
Private Sub TreeView1_Expand(ByVal Mode As Mode)
   Dim X As Integer, ThisDay As String, Month, Year As String, _
      ThisDate As String, itmX As Listltem, MonthStr As String
   If Mode.Key = "" Then Exit Sub 'Do nothing
   Month = Format(Mode.Key, "m")
   MonthStr = Format(Mode.Key, "mmmm")
   Year = Format(Node.Key, "yyyy")

   X = 1
   Do While X <= 31
      ThisDate = Month + "/" + Str(X) + "/" + Year
      If Not IsDate(ThisDate) Then
         X = X + 1
      Else
         ThisDay = Format(ThisDate, "dddd")
         Set itmX = ListView1.Listltems.Add(, , ThisDay, 7, 4)
          itmX.SubItems(1) = MonthStr
          itmX.SubItems(2) = ThisDate
          itmX.SubItems(3) = Year
          itmX.SubItems(4) = ""
         X = X + 1
      End If
   Loop
End Sub
```

Tutto questo svolge bene il compito di aggiungere voci e Subltems a ListView1 (si veda la Figura 8.17). È ovvio che il codice può consentire una grande flessibilità di elaborazione di Listltems e dei corrispondenti SubItems.

Questo programma contiene solo alcune funzioni aggiuntive (anche se si potrebbe farlo diventare un programma per un sofisticato calendario personale). In primo luogo vediamo la codifica che attiva la scelta runtime della proprietà ListView1.View (sistemata in corrispondenza dell'evento clic dell'array dei pulsanti di opzione):

```
Private Sub optLV_Click(Index As Integer)
   ListView1.View = Index
End Sub
```

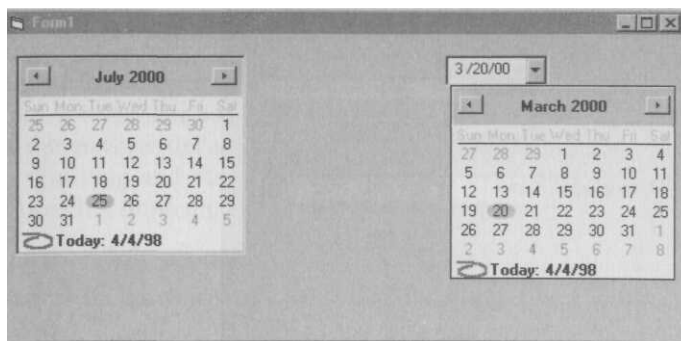
In secondo luogo, quando ListView1 è in modalità *Report*, è previsto un campo Memo che inizialmente non contiene nulla per ogni giorno del mese. Supponiamo di voler inserire e riprendere del testo da mettere nel campo Memo; ho appositamente inserito del codice, appena abbozzato, in corrispondenza dell'evento doppio clic di ListView1. La procedura apre un testo InputBox. Se non esiste testo nel campo Memo, inserisce una stringa predefinita, altrimenti utilizza la stringa esistente. Quando l'utente fa clic, il testo in InputBox viene memorizzato nel campo *Memo* (si veda la Figura 8.18):

```
Private Sub ListView1_DblClick()
Dim MemoValue, DfStr As String
   If ListView1.SelectedItem.SubItems(4) = "" Then
      DfStr = "Bulgy Bears Forever"
   Else
      DfStr = ListView1.SelectedItem.SubItems(4)
   End If
   MemoValue = InputBox("Enter Memo Text", "List View Demo", DfStr)
   ListView1.SelectedItem.SubItems(4) = MemoValue
End Sub
```

Si accede ovviamente all'informazione richiesta da questa procedura mediante la proprietà .SelectedItem di ListView1.

I controlli sul calendario

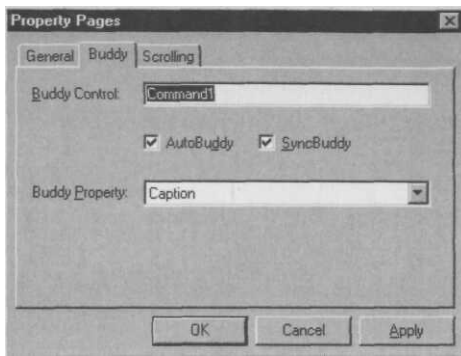
Già che ci stiamo occupando di anni e di date, vediamo due nuovi controlli di VB6 che facilitano la creazione di interfacce che riguardano il tempo. Il controllo MonthView *consente di creare applicazioni che permettono all'utente di visualizzare e definire informazioni sulle date mediante un calendario. Il controllo* DateTimePicker, *detto anche* DTPicker, *consente di predisForre un campo data formattato che permette agli utenti di selezionare facilmente una data. Gli utenti possono selezionare una data dal calendario* MonthView *a discesa collegato a* DTPicker. *La Figura 8.23 mostra questi due controlli (*MonthView *è sulla sinistra, mentre sulla destra è visibile* DTPicker *collegato a* MonthView).

Figura 8.23
Si possono utilizzare i controlli MonthView eDTPicker per creare facilmente interfacce utente per le date.

Creazione di un selettore

Il controllo UpDown è costituito da una coppia di frecce, sulle quali l'utente fa che per incrementare oppure decrementare un valore relativo a un controllo associato identificato come controllo "buddy" ("compagno"). Quando un controllo UpDown viene collegato al suo buddy, i due controlli diventano per l'utente un unico con trollo ibrido.

Si utilizza il controllo UpDown *alposto del controllo* Spin Button *che veniva distribu ito con leprecedenti versioni di Visual Basic.*

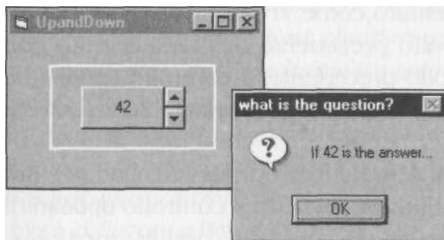
Prima di utilizzare il controllo UpDown all'interno di un progetto occorre aggiungere la libreria Microsoft Windows Common Controls-2 (Mscomt2.Ocx) nella proprio Toolbox. Gran parte dei controlli sulle finestre che visualizzano dati può essere co legata alla proprietà Buddy del controllo UpDown. A questo proposito di solito si uti lizzano pulsanti di comando e caselle di testo ma, dato che il controllo intrinseco e un'etichetta non è una vera e propria finestra, questi non possono essere utilizza come controllo di tipo buddy.
La scheda *Buddy* della *Property Pages* del controllo UpDown, mostrata in Figura *8.2* viene utilizzata per definire un controllo buddy. La si può anche utilizzare per defi nire le proprietà del controllo buddy da collegare al controllo UpDown.
Se AutoBuddy è attivo (oppure definito come True nel codice), il controllo UpDown utilizza automaticamente il controllo precedente dell'ordinamento come suo con trollo buddy. Se non esiste controllo precedente, il controllo UpDown utilizza quello successivo. In alternativa è possibile utilizzare la proprietà BuddyControl di UpDown per assegnare un controllo buddy.
In fase di progettazione, una volta definite le proprietà AutoBuddy e BuddyControl il controllo buddy si associa automaticamente con il controllo UpDown dimensionan dosi e posizionandosi in prossimità di questo (i buddy appaiono uno vicino all'altro). Si può utilizzare la proprietà *Alignment* per sistemare il controllo UpDown destra oppure a sinistra del suo compagno.

Figura 8.24
Si può utilizzare Property Pages del controllo UpDown per assegnare un controllo buddy e leproprietà corrispondenti.

Ilprogettopresente nel CD-ROM allegato al libro con il nome Updown.Vbp combina un controllo UpDown *con unpulsante di comando buddy. La proprietà* SyncBuddy *di* UpDown *è stata definita come* True *e la proprietà* Buddy *collegata è stata definita nella proprietà* Caption *delpulsante di comando (si veda la Figura 8.24).*

Si deve controllare che la didascalia del pulsante di comando sia definita come 40, valore identico a quello della proprietà *Value* (o punto di partenza) del controllo UpDown. Quando l'utente fa clic sulla freccia rivolta verso l'alto il numero visualizzato dalla didascalia del pulsante *Command* cresce; quando fa clic sulla freccia verso il basso, il numero diminuisce. Ho inserito nell'evento Change del controllo UpDown questo codice:

```
Private Sub UpDown1_Change()
   If UpDownl.Value = 42 Then
      MsgBox "If 42 is the answer...", vbQuestion,
         "what is the question?"
   End If
End Sub
```

Quando l'utente fa clic sulla freccia rivolta verso l'alto e il valore della didascalia del pulsante di comando (quindi la proprietà Value di UpDown) arriva al valore 42, viene visualizzato un messaggio come quello visibile in Figura 8.25.

Figura 8.25
Si possono sincronizzare i controlli mediante la proprietà Buddy.

SysInfo

È possibile utilizzare il controllo SysInfo per definire la risposta a modifiche dell'ambiente di sistema. Per esempio, SysInfo fa partire un evento nel caso in cui si modifica la dimensione della schermata oppure quando si collega al sistema un dispositivo di tipo Plug and Play (PnP). Il controllo SysInfo è invisibile in fase di esecuzione. I possibili impieghi di questo controllo sono:

- determinazione della piattaforma e della versione del sistema operativo;
- rilevamento della dimensione di desktop e monitor e modifica della risoluzione;
- rilevamento e gestione dei dispositivi Plug and Play (PnP);
- controllo dello stato della batteria e del collegamento di alimentazione.

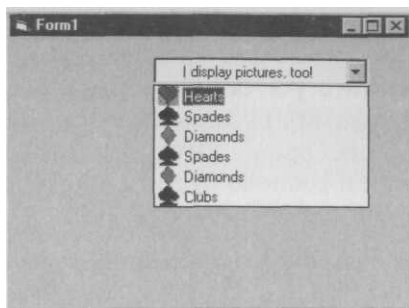
MSFlexGrid

Il controllo MSFlexGrid visualizza e gestisce dati in forma tabellare; dispone di grande flessibilità e consente di ordinare, combinare e formattare tabelle che contengono stringhe e immagini. Per esempio, MSFlexGrid può assumere l'aspetto di un foglio di calcolo. Quando è collegato a un controllo Data, MSFlexGrid visualizza i dati in sola lettura.
In una qualsiasi cella di MSFlexGrid è possibile inserire testo, un'immagine o entrambi gli oggetti; le proprietà Row e Col consentono di specificare la cella corrente di MSFlexGrid. Si può specificare la cella corrente nel codice, ma anche l'utente può modificarla in fase di esecuzione utilizzando il mouse o le frecce di direzione. La proprietà Text fa riferimento al contenuto della cella corrente.
Se il testo di una cella è troppo lungo per essere visualizzato in una sola cella e la proprietà WordWrap è definita come True, il testo prosegue sulla riga successiva all'interno della stessa cella. Per visualizzare questo testo può essere necessario aumentare il valore delle proprietà ColWidth o RowHeight; si utilizzano le proprietà Cols e Rows per definire il numero totale di colonne e di righe del controllo MSFlexGrid.

ImageCombo

Il controllo ImageCombo *è simile alla casella combinata standard di Windows alla quale si aggiunge il fatto che può visualizzare immagini oltre al testo, come mostrato nella Figura 8.26.*

Ogni voce dell'elenco costituisce di per se un oggetto ComboItem. L'insieme dei ComboItem in ImageCombo definisce una collezione ComboItems. (Per avere maggiori informazioni sugli oggetti di una collezione si veda il Capitolo 14.)

Figura 8.26 *Il controllo ImageCombo può essere utilizzato per visualizzare un'immagine in corrispondenza di ciascuna voce del testo contenuto nell'elenco di ImageCombo.*

In modo analogo ad altri controlli già visti in questo capitolo, un ImageCombo gestisce le immagini che utilizza per mezzo di un controllo associato ImageList. Questa associazione può essere definita in fase di progettazione utilizzando il dialogo delle proprietà personalizzato di ImageCombo, altrimenti si può impostarla in fase di esecuzione:

```
Private Sub Form_Load()
    Set ImageCombo1.ImageList = ImageList1
End Sub
```

Le immagini della libreria *ImageList* vengono assegnate alle voci di ImageCombo mediante il numero indice oppure attraverso una stringa assegnata (denominata "chiave"). Il frammento di codice che segue crea un ComboItem e lo aggiunge a ImageCombo mediante la chiave Suits1 ("Clubs" è la stringa di testo da visualizzare):

```
Dim objNewItem As ComboItem
Set objNewItem = ImageCombo1.ComboItems.Add(1,
    "Suits1", "Clubs")
ImageCombo1.ComboItems("Suits1").Image = 1
```

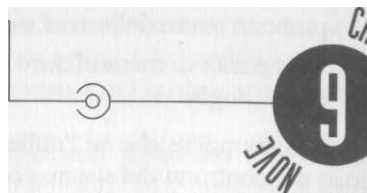
L'ultima riga di codifica assegna la prima immagine della libreria associata *ImageList* alla voce ComboItem appena creata.

Riepilogo

Questo capitolo si è occupato dei controlli ActiveX della Professional Edition che vengono utilizzati per creare gli elementi dell'interfaccia utente di Windows. Alcuni di questi controlli possono ancora sembrare difficili da digerire ma ci si può comunque lavorare sopra, specialmente se si conoscono alcuni trucchi del mestiere (e il mio libro è qui proprio per questo). In sostanza, è facile creare applicazioni che *abbiano l'aspetto* di applicazioni Windows.
Questi controlli OCX non sono solo belli da vedere; quasi senza eccezioni dispongono di potenzialità tali che consentono di aumentare sensibilmente le funzionalità delle vostre applicazioni.

- Avete visto come si creano dialoghi a scheda mediante il controllo SSTab.

- Avete visto come aggiungere menu di scelta rapida e fogli delle proprietà alle applicazioni.
- Avete scoperto come si creano i wizard.
- Vi ho mostrato come si utilizzano i controlli Slider e ProgressBar.
- Vi ho mostrato come si utilizza il controllo RichTextBox per modificare i font all'interno di un controllo di modifica e come aprire, memorizzare e stampare file .Rtf.
- Vi ho spiegato come utilizzare i controlli CoolBar e FlatScrollBar.
- Vi ho spiegato come utilizzare i controlli TreeView e ListView per organizzare e visualizzare informazioni in modo gerarchico.
- Avete imparato a implementare interfacce utente basate su date con i controlli Calendar di VB6.
- Avete visto come associare il controllo UpDown con un "buddy".
- Avete visto come utilizzare i controlli MSFlexGrid e ImageCombo.

USO DEL REGISTRO DI CONFIGURAZIONE

- Logica e finalità del Registro di configurazione
- La persistenza dei file Private Profile di tipo .Ini
- La struttura del Registro di configurazione
- Utilizzo di Regedit.Exe
- Il contenuto del file .Dat del registro
- Unione dei file .Reg
- Registrazione di componenti e controlli ActiveX mediante Regsvr32.Exe e Regocx32.Exe

Questo capitolo illustra le finalità e le funzionalità del Registro di configurazione di Windows (Registry), il quale mette a disposizione un meccanismo centrale di memorizzazione e ricerca delle informazioni riguardanti il sistema e le applicazioni.

Per eseguire Regedit in Windows 95/98, selezionare Esempi *dal menu* Start *di Windows, scrivere* regedit *e fare clic su* OK. *Le modifiche apFortate al registro sono irrevocabili, nel senso che queste modifiche hanno effetto immediato non appena i registro viene chiuso, senza alcun messaggio ulteriore di avvertimento. La modifica di determinate impostazioni del registro, tra le quali per esempio la disattivazione di importanti funzioni del sistema, può avere effetti catastrofici. Si raccomanda quindi difare una copia di riserva del file del registro, selezionando la voce* Export Registry File (Esporta file del Registro di configurazione) *nel menu* Registry *(Registro* di configurazione) *di regedit, prima difare esperimenti con il proprio registro.*

Vantaggi del Registro di configurazione

Il registro centrale di configurazione presenta una serie di elementi che si ripercuotono positivamente sul software applicativo di Windows:

- Una singola locazione per i dati di inizializzazione di un'applicazione. Al contrario, nelle vecchie versioni dei sistemi operativi, come in Windows 3.x, per memorizzare i dati di inizializzazione si utilizzavano file di stringhe di profilo (i file .Ini). Venivano scaricati su disco fisso molti file .Ini (di

solito nella directory Windows oppure nella directory di avvio dell'applicazione) e non era sempre possibile risalire a quale applicazione avesse trasferito un certo file .Ini. (Inoltre, alcune applicazioni scrivevano e riprendevano informazioni dai file di profilo di tipo pubblico, Win.ini e System.ini).

- La capacità di annidare le informazioni; in altre parole le voci del registro possono avere delle voci subordinate.
- La capacità di memorizzare e ricercare valori di tipo binario, oltre alle semplici stringhe.

Si può aggiungere che se l'utilizzo di un Registro di configurazione risulta vantaggioso nei confronti del sistema operativo, questo fatto si riversa positivamente sugli utenti stessi; anche chi scrive software ha vita più facile. Gli aspetti favorevoli riguardano:

- Una singola sorgente di dati per elencare e configurare le impostazioni hardware, software, quelle relative ai driver dei dispositivi e del sistema operativo.
- Un semplice metodo di ripristino delle informazioni nel caso di avaria del sistema. Il sistema è in grado di ritornare automaticamente all'ultima configurazione funzionante (quella che comprende le impostazioni del registro che hanno avviato con successo il computer e il sistema operativo di Windows).
- Una migliore possibilità di configurare le impostazioni, da parte di utenti e amministratori, mediante gli strumenti del Pannello di controllo e altri strumenti di amministrazione, senza la modifica diretta dei file di configurazione, riducendo quindi la possibilità di introdurre degli errori.
- La possibilità di utilizzare un insieme di funzioni indipendenti dalla rete per definire e ricercare dati remoti relativi alla configurazione in rete, il che consente una più facile amministrazione del sistema.
- La possibilità di mantenere preferenze utente e autorizzazioni di accesso multiple su una sola macchina.

La permanenza in vita delle stringhe di profilo private (i file .Ini)

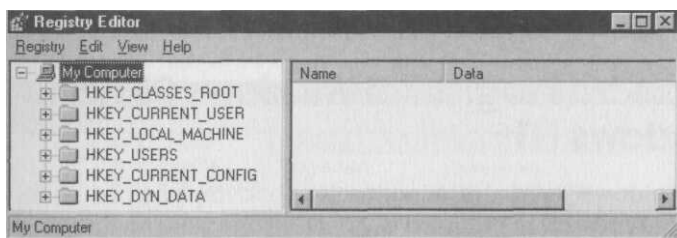
Andrebbe sempre utilizzato il Registro di configurazione per memorizzare e ricercare le informazioni di inizializzazione; questo è in effetti uno dei principali requisiti per ottenere il riconoscimento ufficiale di compatibilita Windows. È anche una regola dettata dal buon senso pratico, dato che le voci del registro sono facili da usare, più flessibili e affidabili rispetto ai file .Ini. (Si veda il Capitolo 10 per avere informazioni complete su come si programma con il registro.)

Nonostante tutto questo i file .Ini non sono scomparsi del tutto (anche se forse non è vero che "i file .Ini saranno sempre tra noi!"). I file .Ini pubblici, Win.ini e System.ini sono stati lasciati per motivi di compatibilita pregressa; in parole povere,

a1cu ne applicazioni a 16 bit ancora in uso non possono funzionare se non leggono e scrivono su questi file. È ancora previsto il formato .Ini dei file e, sotto le API a 32 bit, è possibile trovare versioni aggiornate delle funzioni ReadPrivateProfile-String e WritePrivateProfileString che venivano utilizzate nei giorni lontani di 3 1. In definitiva, per una ragione o per un'altra si deve ancora fare i conti con applicazioni che utilizzano i file di profilo per conservare le informazioni che le riguardano. Questo significa che, se si vuole, è ancora lecito utilizzare i file .Ini nelle proprie applicazioni anche se, nonostante alcune eccezioni (che vedremo tra un momento), sarebbe meglio evitarlo. L'utilizzo del registro è più facile e si ottengono risultati migliori.
In particolare, alcune applicazioni Windows di Microsoft utilizzano file .Ini, in aggiunta oppure al posto del registro (confermando il motto "Fate quello che dico, non fate quello che faccio"); per esempio, Telephony API (l'API) utilizza Tele-phon.Ini. Anche il nostro beniamino Visual Basic 6 fa uso di file .Ini. Si utilizza per esempio Vb.Ini per le informazioni di controllo degli aggiornamenti CLSID (vedremo qualcosa in più su CLSID più avanti in questo capitolo); Vbassin.ini, un file nuovo per la versione 6 di VB, serve a registrare un add-in in VB (si veda a questo proposito il Capitolo 29).

La struttura del Registro

Il Registro di configurazione di Windows è costituito da una struttura gerarchica suddivisa in sei sottoalberi. Come si vedrà più avanti in questo capitolo, è possibile utilizzare l'Editor del registro Regedit.Exe, mostrato in Figura 9.1, per visualizzare graficamente questa struttura.

Figura 9.1 *L'editar del registro (Regedit.Exe) mostra i sei rami principali del Registro di configurazione di Windows.*

Ogni intestazione di ramo inizia formalmente con la parola chiave HKEY, per esem-pio HKEY_CURRENT_USER. Si utilizza la parola chiave KEY in quanto ogni ramo gestisce un insieme differente di valori chiave.

Gerarchia del Registro

Vediamo una descrizione dei sei sottoalberi (un "sottoalbero" definisce una struttura costituita da chiavi e sotto chiavi) e il genere di informazioni che contengono:

- HKEY_CLASSES_ROOT. Questo ramo contiene informazioni sulle estensioni dei file, associazioni tra file e applicazioni che supportano il drag and drop,

dati OLE e le informazioni che riguardano le scorciatoie di Windows 95 (che sono, in effetti, collegamenti OLE). HKEY_CLASSES_ROOT è una copia aggiornata in tempo reale (o alias) di HKEY_LOCAL_MACHINE\Software\Classes.

- HKEY_CURRENT_USER. La sezione di HKEY_USERS che fa riferimento all'utente attuale. Se è disponibile un solo utente, HKEY_USERS e HKEY_CURRENT_USER sono identici tra loro.
- HKEY_LOCAL_MACHINE. Il computer e l'hardware installato; sono possibili configurazioni multiple, che vengono aggiornate dinamicamente.
- HKEY_USERS. Contiene informazioni su scrivania, rete e dati particolari dell'utente. Questi dati sono memorizzati nel file User.Dat.
- HKEY_CURRENT_CONFIG. Questo ramo contiene le impostazioni relative al monitor e alle stampanti disponibili.
- HKEY_DYN_DATA. Questo ramo memorizza informazioni relative alle prestazioni di Windows; è possibile analizzare questi dati utilizzando le applicazioni System Monitor.

Dal punto di vista del software di installazione, si deve lavorare con il sottoalbero del software in HKEY_LOCAL_MACHINE (si è visto che il sottoalbero di classi relativo a questo ramo viene copiato in HKEY_CLASSES_ROOT). Le informazioni specifiche sull'utente che riguardano la configurazione di un'applicazione sono memorizzate in HKEY_USERS nella stessa posizione relativa delle informazioni su quel software in HKEY_LOCAL_MACHINE\Software\Description.
Può anche essere necessario utilizzare HKEY_LOCAL_MACHINE per ricavare informazioni sull'hardware della macchina di destinazione e HKEY_USERS per avere altre informazioni particolari, tra le quali il nome dell'utente, il nome della società, il numero di telefono e così via.

Differenze tra i registri di Windows 95/98 e di Windows NT

I registri di Windows 95/98 e di Windows NT sono implementati in modi differenti. Alcune funzioni presenti nel registro di NT non sono state incluse in quello di Windows 95/98.

La differenza fondamentale tra i due registri che gli sviluppatori devono conoscere riguarda il fatto che il registro di Windows 95/98 non prevede attributi di protezione e non può quindi essere considerato sicuro. Il registro di Windows NT, d'altro canto, è stato progettato tenendo ben presenti le considerazioni relative alla sicurezza.

Si possono notare altre differenze significative. Il registro di Windows 95/98 non prende il posto di Config.sys, Autoexec.Bat, Win.ini, System.ini e dei gruppi Program Manager. I vecchi programmi possono continuare ad utilizzare questi file di inizializzazione e le corrispondenti tecniche di configurazione.

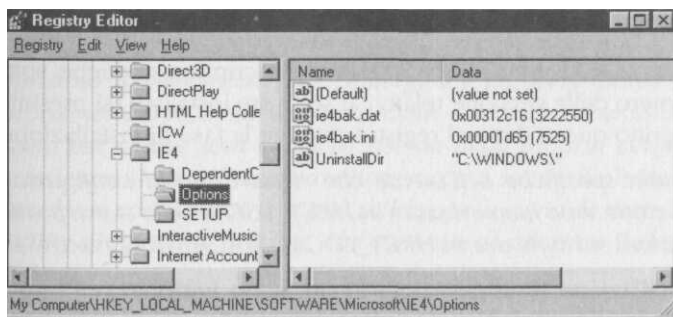
Con Windows NT, invece, i dati di configurazione del sistema che potrebbero tro-are posto in file pubblici .Ini vengono automaticamente sistemati nel Registro di epurazione. I file Win.ini e System.ini esistono ancora esclusivamente per far funzionare le applicazioni a 16 bit.
Infine si deve considerare che alcune API del registro, per esempio RegOpenKeyEx, non si comFortano allo stesso modo con Windows 95/98 e con NT. L'utilizzo delle API del registro è trattato nel Capitolo 10. La risorsa migliore per studiare i problemi di compatibilita del sistema operativo nei confronti di particolari funzioni API è Win32 SDK Knowledge Base nella libreria MSDN.

Parole chiave

Le voci del registro sono scritte nella forma "parola chiave contiene valore". Le parole chiave possono includere delle sottochiavi e sono scritte in un modo che dipende dal tipo di dati dei loro valori. Windows 95 e NT utilizzano attualmente tre tipi di valori nelle parole chiave:

- Binario. Per esempio, le informazioni sull'hardware sono per lo più memorizzate come dati binari che possono essere visualizzati in Regedit in formato binario oppure esadecimale.
- Testo. Una stringa di testo, per esempio il messaggio di avvio del mio computer: "Frodo says Hi!".
- DWord. Un intero senza segno a 32 bit oppure l'indirizzo di un segmento e del suo offset associato. DWord è un tipo di dati comunemente utilizzato in Windows SDK e in C++.

Il tipo di dati di una chiave è identificato in Regedit dalla sua icona (si veda la Figura 9.2). I valori DWord utilizzano l'icona del tipo dati binario.

Figura 9.2 *Valori delle chiavi visualizzati in Regedit (si possono notare le icone dei tipi dati binario e testo).*

La dimensione di un valore non può essere superiore a 64 K, mentre il limite del numero di voci del registro dipende dallo spazio disponibile su disco fisso.

Il sottoalbero del software in HKEY_LOCAL_MACHINE

Il sottoalbero HKEY_LOCAL_MACHINE contiene le informazioni del registro di Windows 95 che riguardano la configurazione e l'inizializzazione di tutto il software installato. Le voci di questo ramo si applicano a tutti quelli che utilizzano il computer. Qui vengono incluse anche le informazioni sulle associazioni dei file e su OLE.

Classi

Il sottoalbero HKEY_LOCAL_MACHINE\Software\Classes definisce i tipi di documenti e fornisce le informazioni sull'associazione OLE e sulle estensioni dei nomi di file che possono essere utilizzate dalle applicazioni (per esempio, nelle operazioni di drag and drop). Si è visto che HKEY_CLASSES_ROOT è un alias di questo sottoalbero. In effetti HKEY_CLASSES_ROOT non fa altro che puntare a HKEY_LOCAL_-MACHINE\Software\Classes allo scopo di garantire la compatibilita con il database di registrazione di Windows 3.x. Il sottoalbero Classes contiene due tipi di sottovoci:

- Sottovoci relative alle estensioni dei nomi di file, che specificano la definizione della classe associata con i file di una determinata estensione.
- Sotto voci di definizione della classe, che specificano le proprietà OLE e della shell di una classe (tipo) di documento.

// sottoalbero CLSID relativo a un componente o un controllo ActiveX elencato nella sezione Classes contiene un valore estremamente imFortante, il Class ID del server, altrimenti detto CLSID. Un CLSID è un numero esadecimale, generato durante la creazione del server OLE (oggetto ActiveX), che identifica in maniera univoca un server OLE. Il CLSID, insieme all'equivalente leggibile che viene definito nel registro, serve ad attivare l'oggetto.

Descrizione

Il ramo HKEY_LOCAL_MACHINE\Software\Description contiene sottovoci con il nome e il numero della versione relativi al software installato (si presume che il software abbia scritto questi dati nel registro durante la fase di installazione).

Le informazioni specifiche dell'utente che riguardano la configurazione di una cena applicazione sono memorizzate in HKEY_USERS nella stessa posizione relativa alle informazioni sul software in HKEY_LOCAL_MACHINE\Software\Description.

Durante l'installazione le applicazioni aggiungono informazioni in Software nella forma seguente:

HKEY_LOCAL_MACHINE\Software\CompanyName\ProductName\Version

La sottovoce denominata

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion

contiene sottovoci che includono informazioni sulla configurazione del software che fa parte integrante del sistema operativo di Windows.

Utilizzo di Regedit

Regedit rappresenta la strada più semplice per esaminare la struttura del registro, le sue voci e i valori corrispondenti (si vedano le Figure 9.1 e 9.2). Regedit può anche essere utilizzato per aggiungere o cancellare delle voci e per modificarne manualmente i valori. Se vi accorgete di utilizzare spesso Regedit, vi conviene aggiungere un collegamento sulla scrivania oppure lo potete inserire nel menu *Start* di Windows.

Attenzione! Regedit.Exe non prevedeforme di controllo che consentano di nonfare modifiche inopFortune. Una voltafatta una modifica e usciti da Regedit, non c'èpiù modo di tornare indietro! Non esiste alcun messaggio "Vuoi salvare le modifiche?", non c'è niente di tutto questo.

Di conseguenza è meglio non fare modifiche manuali se non si è proprio sicuri del significato di una chiave e dei valori che può assumere; inoltre, si seguano le procedure indicate nella guida online di Regedit per essere sicuri che venga effettuata una copia di riserva del registro prima di apFortare modifiche.

Èpossibile modificare il registro anche dalprompt del DOS; questo modo di operare può risultare comodo nel caso in cui il registro sia danneggiato e risulti impossibile avviare Windows. Per avere ilprompt del DOS senza avviare Windows 95/98, premere F8 durante la notifica dell'avvio di Windows. Quando compare ilprompt del DOS, scrivere Regedit /? *per avere istruzioni su come utilizzare Regedit da DOS.*

Riparazione di registri danneggiati

Può essere che qualcuno non sappia che Windows 95/98 vienefornito con una utility per il ripristino d'emergenza ERU (Emergency Recovery Utility); questo programma si trova nel CD-ROM di Windows nella directory Other\Misc\Eru, e può essere utilizzato per creare una copia di riserva della propria configurazione di sistema. Successivamente sipuò sistemare questa copia su un dischetto di avvio. Se si verifica unproblema con il registro, sipuò utilizzare la copia di riserva in combinazione con l'utility di ripristino Erd.Exe, che consente di riFortare il sistema alle condizioni precedenti.

In altre parole l'utility ERU consente di fare una copia del registro, che viene ripristinata da Erd.Exe. Come in tutte le situazioni di questo tipo, l'operazione di ripristino non può funzionare se prima non è stata fatta una copia di riserva.

Modifica dei valori nelle parole chiave del registro

Si possono modificare i valori del registro selezionando *Modify* dal menu *Edit* di Regedit oppure facendo clic destro sulla parola chiave interessata. In entrambi i casi si apre un dialogo *Edit,* che dipende dal tipo di dato contenuto nella chiave e fa riferimento alla parola chiave corrente (si vedano le Figure da 9-3 a 9.5). Si può quindi utilizzare il dialogo *Edit* per modificare il valore.

Figura 9.3
Regedit utilizza il dialogo Edit Binary Value per modificare un valore binario presente nel registro.

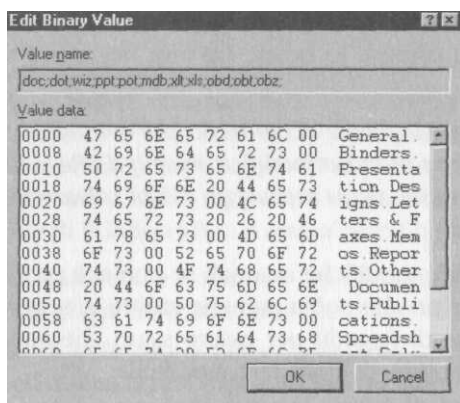



Figura 9.4
Regedit utilizza il dialogo Edit Strìngper modificare un valore di stringa presente nel registro.

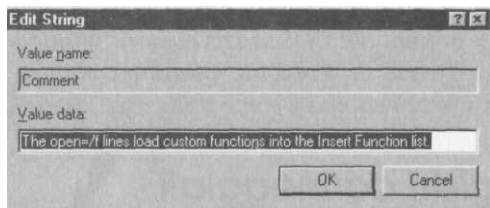
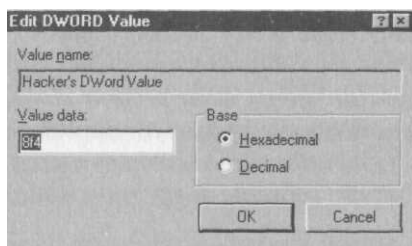



Figura 9.5
Regedit utilizza il dialogo Edit DWORD Value per modificare un valore DWord presente nel registro.

Inserimento e cancellazione di parole chiave

Per aggiungere una sottovoce, selezionare il genitore della sottovoce interessata nel pannello di sinistra di Regedit e scegliere *New,* facendo clic destro oppure utilizzando il menu *Edit.* In entrambi i casi si ha la possibilità di selezionare il tipo di chiave, come mostrato in Figura 9.6.

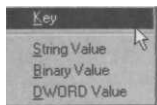
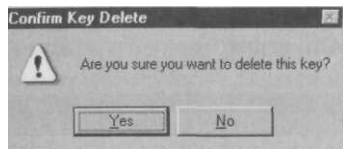
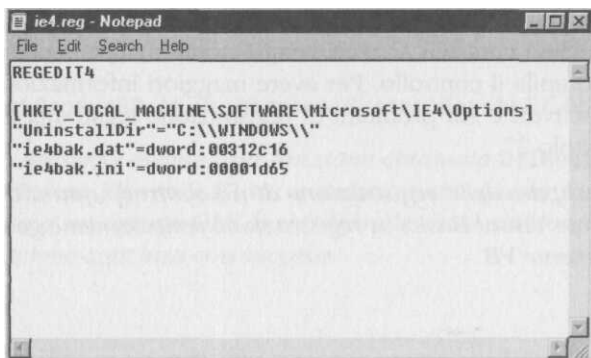
Figura 9.6
I valori che può contenere una chiave.

Per cancellare una parola chiave, selezionarla nel pannello di destra di Regedit con un clic destro oppure utilizzando il menu *Edit,* quindi scegliere *Delete.* Compare il messaggio mostrato in Figura 9.7; scegliere Yes per confermare la cancellazione.

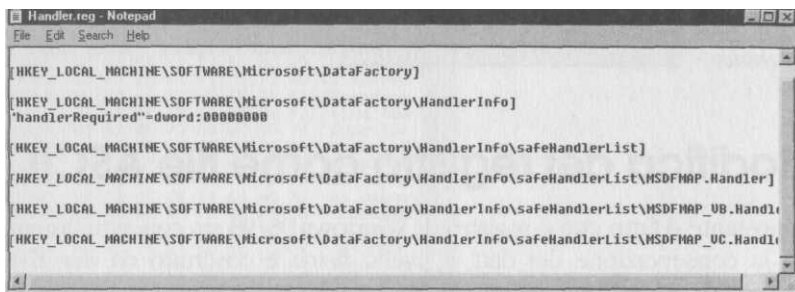
Figura 9.7
Questo è il solo messaggio che vi trovate a leggere prima di dire addio alla chiave.

Modifica del registro come file ASCII

Nonostante il fatto che il registro di Windows 95/98 sia concettualmente un sistemi per la conservazione dei dati, a livello fisico è costituito da due file: User.Dat e System.Dat. User.Dat contiene le informazioni che riguardano i profili utente e 1a diverse configurazioni; System.Dat contiene le impostazioni delle specifiche hard ware e del computer. Device Manager costituisce l'interfaccia grafica principale pe apFortare modifiche al contenuto di System.Dat.
User.Dat e System.Dat sono file di tipo binario e come tali possono essere visualiz zati e modificati con un editor in grado di elaborare file binari. Il contenuto del regi stro può però essere esFortato anche come file ASCIl; il file esFortato ha estensiono .Reg. È possibile esFortare l'intero contenuto del registro oppure quello relativo a un solo ramo.
Si può quindi utilizzare un editor ASCIl per modificare il file .Reg. Per esempio, la Figura 9.8 mostra un file .Reg in WordPad. Una volta effettuate le modifiche, si può imFortare il file .Reg modificato nel registro. Per esFortare il registro, selezionare *ExFort Registry File* dal menu *Registry.* Per imFortarlo, selezionare *ImFort Registr File,* sempre dal menu *Registry.*

Figura 9.8
Questo è un ramo del file .Reg visualizzato in WordPad.

Si consiglia di evitare l'opzione Print *del menu* Registry. *Ci vuole molto tempo per stampare un registro di Windows, rimangono sicuramente esclusi molti prodotti e il risultato che si ottiene non è poi così maneggevole.*

Combinazione di file .Reg del registro

Spesso le applicazioni vengono distribuite con i propri file ASCII .Reg. Questi file contengono le chiavi e i valori che si devono aggiungere al registro; la Figura 9.9 mostra le voci che riguardano un tipico file di registro.

Figura 9.9 *Handler.Reg visualizzato in Notepad mostra la Forzione del file di registro ASCIl relativa a DataFactory di Microsoft.*

Per combinare uno di questi file .Reg all'interno del registro, fare doppio clic sul file oppure clic destro sul file e poi scegliere *Merge.*

Registrazione di componenti e controlli ActiveX

Si possono inserire componenti e controlli ActiveX nel registri di Windows in molti modi. In particolare i componenti ActiveX di Visual Basic (server OLE) vengono registrati sul vostro sistema quando sono compilati. Il wizard Package and Development facilita 'a registrazione di questi oggetti nei sistemi dei vostri utenti.

Per avere maggiori informazioni sulla registrazione (e sulla cancellazione dal registro) delle applicazioni ActiveX scritte da voi, consultate il Capitolo 20 e il Capitolo 23. Per conoscere il wizard Package and Development vedete il Capitolo 35.

In modo analogo, anche i controlli ActiveX vengono automaticamente registrati nel sistema quando si compila il controllo. Per avere maggiori informazioni sulla creazione dei controlli ActiveX e sui problemi di registrazione legati alla loro distribuzione, si veda il Capitolo 27.

Non ci si deve preoccupare della registrazione di un controllo quando si effettua il suo debug in ambiente Visual Basic; la registrazione temForanea è gestita in modo automatico dall'ambiente VB.

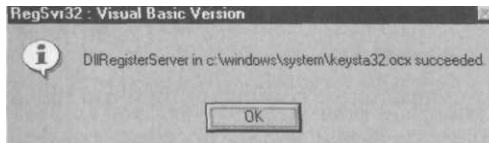
Nonostante questo, cosa si deve fare per registrare sul proprio sistema un componente o un controllo ActiveX che sia stato compilato da qualcun altro (e non viene fornito di routine di installazione)? Oppure, cosa si deve fare per registrare un server (o un controllo) sul sistema di qualcun altro senza preoccuparsi di predisForre un programma di installazione?
Questi compiti sono svolti da tre utility che sono distribuite con VB6: Regsvr32.Exe, Regocx32.Exe e Regit.Exe; questi programmi si trovano sul primo CD-ROM Visual Studio nella directory \Common\Tools\Vb\Regutils. I file indicati devono essere copiati sul disco fisso.

Se sipensa di utilizzare spesso queste utility, conviene aggiungere la directory che le contiene tra i percorsi preferiti. Si possono anche associare le estensioni .Dll e' .Ocx con Regsvr32.Exe, in modo che si possano registrare oggetti ActiveX facendo doppio clic su di essi.

Si utilizza Regsvr32.Exe per registrare manualmente (e per togliere la registrazione) di componenti e controlli ActiveX (OCX). Al programma viene passato il nome di file di un oggetto OLE (server di un controllo ActiveX) da registrare come parametro su riga di comando. Se il nome del file è preceduto dall'opzione /u, il programma toglie la registrazione relativa all'oggetto OLE, invece di aggiungerla. Regsvr32 riForta un messaggio che segnala l'avvenuta operazione. Per esempio, se Regsvr32.Exe è stato copiato nella directory C:\Vb\Tools\, se si esegue

```
C:\Vb\Tools\Regsvr32 C:\Windows\System\Keysta32.0cx
```

si chiede di aggiungere Keysta32.Ocx nel registro. Regsvr32 riForta il messaggio mostrato in Figura 9.10 nel caso in cui la registrazione si concluda con successo.

Figura 9.10 *Keysta32.Ocx è stato registrato con successo.*

È possibile disattivare il messaggio di risposta se si esegue Regsvr32.Exe con l'opzione /s.

Per togliere la registrazione relativa all'oggetto precedente (lo si rimuove così dal registro), eseguire Regsvr32 con l'opzione /u:

```
C:\Vb\Tools\Regsvr32 /u C:\Windows\System\Keysta32.0cx
```

In sostanza, Regsvr32 esegue una funzione chiamata DllRegisterServer. *// messaggio mostrato in Figura 9.10 sta a indicare che questa funzione, responsabile della richiesta a un oggetto OLE di registrare le classi che contiene, ha riFortato un flag di operazione conclusa con successo.*

Registrazione di OCX mediante Regocx32.Exe

Regocx32.Exe lavora in modo simile a Regsvr32.Exe, tranne per il fatto che si applica solo a controlli ActiveX (OCX) e che non riForta un messaggio per indicare l'avvenuta o meno operazione.

Regocx32 è stato studiato specificatamente per il suo utilizzo nei programmi di installazione; questo è il motivo per cui non visualizza alcun dialogo durante la registrazione dei controlli ActiveX.

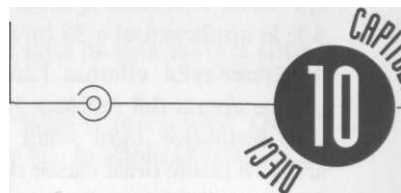
Regit.Exe

*Regit è un'utility a riga di comando che può essere utilizzata con caratteri jolly per registrare diversi oggetti ActiveX. Per esempio, Regit.Exe *.Qcx registra tutti i file .Ocx contenuti nella directory.*

Riepilogo

Questo capitolo ha trattato quello che occorre sapere sulla logica e sulla struttura del registro di Windows e dovrebbe consentirvi la modifica manuale delle voci contenute nel registro. Queste informazioni possono essere considerate come il materiale necessario per la programmazione del registro. Occorre esserne ben consapevoli prima di affrontare la vera e propria programmazione del registro in Visual Basic, argomento del prossimo capitolo.

- Avete visto perché esistono ancora i file .Ini (e quando utilizzarli).
- Avete compreso la struttura del registro.
- Avete imparato a utilizzare Regedit.Exe.
- Avete imparato a esFortare, imFortare e combinare fra loro i file .Reg.
- Avete imparato ad aggiungere, modificare e cancellare parole chiave e valori.
- Ho trattato la registrazione manuale di componenti e controlli ActiveX utilizzando Regsvr32.Exe, Regocx32.Exe e Regit.Exe.

PROGRAMMAZIONE DEL REGISTRO

- API del registro
- Dichiarazioni richieste per l'API e strutture collegate
- Le istruzioni del registro incorForate in Visual Basic
- Chiavi, sottochiavi, rami e valori di ricerca nel registro; inserimento ed eliminazione di nuove chiavi e valori
- Creazione di un componente ActiveX per incapsulare funzioni del registro
- Registrazione dell'estensione di un file

Questo capitolo illustra quello che si deve sapere per utilizzare correttamente il registro nei propri programmi VB6.

API del registro

Esaminiamo un elenco delle API collegate al registro con una breve descrizione di quello che possono fare. Per avere maggiori informazioni su una API specifica, consultare la corrispondente sezione Platform SDK di MSDN. Si possono trovare altre informazioni sull'utilizzo di alcune API del registro più avanti in questo capitolo.

- RegCloseKey rilascia il gestore di una specifica chiave, liberando quindi le risorse.
- RegConnectRegistry stabilisce un collegamento con il gestore del registro predefinito su un altro computer, per esempio attraverso una rete. Richiama RegCloseKey una volta stabilito il collegamento.
- RegCreateKey genera una chiave determinata dalla chiamata della funzione. Se la chiave esiste già nel registro, la funzione la apre. Questa funzione prevede la compatibilita con Windows versione 3.1; le applicazioni a 32 bit utilizzano invece RegCreateKeyEx.
- RegCreateKeyEx genera la chiave stabilita dalla funzione. Se la chiave esiste già nel registro, la funzione la apre; la chiave generata dalla funzione RegCreateKeyEx non contiene valore. Il valore può essere definito mediante le funzioni RegSetValue o RegSetValueEx.

- RegDeleteKey elimina una chiave e tutti i rami e le sottovoci contenuti in essa.
- RegDeleteValue elimina un valore dal registro.
- RegEnumKey effettua l'enumerazione delle sottovoci contenute in una chiave aperta del registro. La funzione riprende il nome della sotto voce ogni volta che viene chiamata. Questa funzione prevede la compatibilita con Windows 3.1; le applicazioni a 32 bit utilizzano invece RegEnumKeyEx.
- RegEnumKeyEx effettua l'enumerazione delle sottovoci contenute in una chiave aperta del registro. La funzione recupera le informazioni riguardanti una sottovoce ogni volta che viene chiamata. RegEnumKeyEx recupera anche il nome della classe della sotto voce e il momento in cui è stata modificata per l'ultima volta.
- RegEnumValue effettua l'enumerazione dei valori di una chiave aperta del registro. Ogni volta che viene chiamata la funzione copia il nome di un valore indicizzato e un blocco di dati relativi alla chiave.
- RegFlushKey scrive tutti gli attributi di una chiave aperta nei file dati relativi al registro su disco fisso.
- RegGetKeySecurity recupera una copia della struttura di sicurezza che protegge una chiave aperta del registro (si applica nel caso di Windows NT).
- RegisterClass registra una classe di finestre per un uso successivo (la classe poi deve essere creata!). In teoria è possibile utilizzare le API pertinenti per registrare, creare e utilizzare una nuova classe di finestre in VB; tuttavia, di solito Visual C++ rappresenta un linguaggio di sviluppo più idoneo per svolgere queste operazioni. Questa funzione prevede la compatibilita con Windows 3.1; le applicazioni a 32 bit utilizzano invece RegisterClassEx.
- RegisterClassEx è la versione di RegisterClass da utilizzare con Windows 95/98 e NT. Si veda la descrizione di RegisterClass.
- RegisterClipboardFormat registra un nuovo formato degli Appunti.
- RegisterEventSource restituisce il gestore da utilizzare con la funzione ReFortEvent per memorizzare un evento, in modo che compaia nell'applicazione Event Viewer. RegisterEventSource deve essere chiamata con un nome sorgente che sia una sottovoce di un file log contenuto nella chiave EventLog del registro.
- RegisterHotKey definisce un tasto di scelta rapida *(hotkey).*
- RegisterWindowMessage definisce un nuovo messaggio di Windows, univoco per tutto il sistema. Il valore del nuovo messaggio può essere utilizzato chiamando le funzioni SendMessage o PostMessage e consente la comunicazione tra applicazioni che cooperano. Per avere maggiori informazioni si veda il Capitolo 11.

- RegLoadKey genera una sottovoce in corrispondenza di HKEY_USER o HKEY_LOCAL_MACHINE e memorizza in quella sottovoce le informazioni di registrazione da un file "hive".

Un "hive" definisce un insieme di chiavi, sotto voci e valori che ha la sua radice in cima alla gerarchla del registro; un hive viene memorizzato nel formato ASCIl .Reg previsto per il registro.

- RegNotifyChangeKeyValue identifica quando è stata modificata una chiave del registro oppure una delle sue sottochiavi.
- RegOpenKey apre una chiave del registro per ulteriori operazioni. Questa funzione prevede la compatibilita con Windows 3.1; le applicazioni a 32 bit utilizzano invece RegOpenKeyEx.
- RegOpenKeyEx apre la voce indicata del registro per ulteriori operazioni.
- RegQuerylnfoKey recupera informazioni che riguardano una chiave del registro.
- RegQueryValue prevede la compatibilita con Windows 3.1; le applicazioni a 32 bit utilizzano invece RegQueryValueEx.
- RegQueryValueEx recupera il tipo e i dati relativi a un valore associato con la chiave aperta del registro.
- RegReplaceKey sostituisce il file copiando una chiave e tutte le sue sotto voci da un altro file, in modo che, quando il sistema verrà avviato nuovamente, la chiave e le sottochiavi conterranno i valori indicati dal nuovo file.
- RegRestoreKey legge le informazioni del registro contenute in un file ASCII .Reg e le copia nelle chiavi corrispondenti. Le informazioni del registro possono includere una chiave e più livelli di sottochiavi.
- RegSaveKey memorizza la chiave, le sue sottochiavi e i valori in un file.
- RegSetKeySecurity stabilisce la protezione di una chiave aperta del registro utilizzando una variabile di supForto di tipo SECURITY_INFORMATION per i dettagli relativi alle impostazioni di sicurezza.
- RegSetValue prevede la compatibilita con Windows 3.1; le applicazioni a 32 bit utilizzano invece RegSetValueEx.
- RegSetValueEx memorizza i dati nel campo valore di una chiave aperta del registro; può anche definire valori addizionali e informazioni di tipo, sempre relativi alla chiave aperta.
- RegUnLoadKey elimina dal registro una chiave e le sue sottochiavi (il suo *hive).* Si può utilizzare RegUnLoadKey per rimuovere un hive dal registro, ma questo non modifica il file che contiene le informazioni del registro (si veda all'inizio di questa pagina per una definizione di "hive").

Dichiarazioni API

Una descrizione completa delle API del registro e dei tipi collegati richiesti da Visual Basic si può trovare nel modulo RegAPI.Bas; questo modulo si trova nel CD-ROM allegato al libro nella directory relativa ai programmi del Capitolo 10. Per utilizzare queste dichiarazioni nei vostri progetti, è sufficiente copiare il modulo nella directory del progetto e aggiungerla al progetto stesso.

Non è mia intenzione riprendere qui tutte le dichiarazioni e le strutture Visual Basic collegate; mi limito a introdurne alcune per darvi un'idea chiara di che cosa si tratti. Per esaminare la sintassi delle altre dichiarazioni API del registro fate riferimento al modulo appena citato.

Una strada comune per conoscere la sintassi delle API consiste nell'utilizzare l'applicazione API Text Viewer e copiare e incollare le dichiarazioni desiderate nel proprio progetto. Per avere maggiori informazioni su API Text Viewer, si vedano il Capitolo 4 e il Capitolo 11.

Vediamo come vengono definite alcune costanti del registro nel modulo RegAPI.Bas:

```
'Costanti del Registro
Public Const HKEY_CLASSES_ROOT = &H80000000
Public Const HKEY_CURRENT_USER = &H80000001
Public Const HKEY_LOCAL_MACHINE = &H80000002
Public Const HKEY_USERS = &H80000003
```

Il modulo definisce parecchie altre costanti che riguardano argomenti come i diritti di accesso, i codici di errore e altro. Vediamo le definizioni di tipo per le strutture che contengono informazioni sul momento di creazione del file e quelle relative alla protezione dell'applicazione:

```
Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Type SECURITY_DESCRIPTOR
    Revision As Byte
    Sbz1 As Byte
    Control As Long
    Owner As Long
    Group As Long
    Sacl As ACL
    Dacl As ACL
End Type
```

Si può notare che SECURITY_DESCRIPTOR contiene il riferimento ACL a una struttura che è definita a sua volta nel modulo. Vediamo alcune dichiarazioni API (per maggiore chiarezza, in qualche caso ho spezzato su più righe, con i trattini di continuazione, dichiarazioni che sono in realtà su un'unica riga).

```
Declare Function RegEnumKey Lib "advapi32.dll" Alias _
   "RegEnumKeyA" (ByVal hKey As Long, _
   ByVal dwIndex As Long, ByVal lpName As String, _
   ByVal cbName As Long) As Long

Declare Function RegEnumKeyEx Lib "advapi32.dll" Alias _
   "RegEnumKeyExA" (ByVal hKey As Long,
   ByVal dwIndex As Long, ByVal lpName As String, _
   lpcbName As Long, lpReserved As Long, _
   ByVal lpClass As String, lpcbClass As Long, _
   lpftLastWriteTime As FILETIME) As Long

Declare Function RegGetKeySecurity Lib "advapi32.dll" _
   (ByVal hKey As Long, ByVal SecurityInformation As Long, _
   pSecurityDescriptor As SECURITY_DESCRIPTOR, _
   lpcbSecurityDescriptor As Long) As Long

Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias _
   "RegOpenKeyExA" (ByVal hKey As Long, _
   ByVal lpSubKey As String, ByVal ulOptions As Long, _
   ByVal samDesired As Long, phkResult As Long) As Long
```

Non c'è niente di particolare: si tratta di comuni dichiarazioni esterne, facili da usare a patto che vengano indicati i corretti tipi di dati. Si noti che le funzioni API a 32 bit, che avrebbero previsto parametri interi nella loro incarnazione a 16 bit, utilizzano invece il tipo di dati interi lunghi (spesso chiamato semplicemente tipo "lungo"). Inoltre, si deve ricordare che è lungo il tipo di dati relativo a un handle; per esempio, hKey rappresenta normalmente Phandle di una chiave.

Infine, è imFortante ricordare che le variabili stringa alle quali si fa riferimento nelle API sono stringhe C, non stringhe VB; questo significa che sono puntatori a una locazione di memoria che memorizza array di caratteri con terminatore nullo. (Terminatore nullo significa che il carattere finale dell'array è il carattere ASCII zero.)

Per chiamare da VB una funzione C con un parametro stringa, e quindi anche una delle API con un parametro stringa, sono richiesti accorgimenti particolari. Si possono adottare diverse tecniche, come verrà discusso nel Capitolo 11, ma a questo punto è sufficiente vederne una.
Nel programma VB, dichiarate una variabile stringa (nell'esempio, szBuffer) e una variabile lunghezza(lBuffSize):

```
Dim szBuffer As String, lBuffSize As Long
```

Successivamente, utilizzate la funzione Space per assegnare a szBuffer una lunghezza fissa riempita di spazi (ci si deve assicurare che il parametro lunghezza della funzione Space sia maggiore della massima lunghezza di stringa che ci si aspetta di ricevere dalla chiamata API):

```
szBuffer = Space(255)
```

Come ultimo passo, prima di richiamare la funzione API, assegnate a lBuffSize la lunghezza szBuffer:

```
lBuffSize = Len(szBuffer)
```

Infine, si possono utilizzare szBuffer e lBuffSize per chiamare una API che richiede un argomento stringa. Si può accedere al valore di szBuffer oppure lo si può assegnare come se fosse una normale stringa VB. La codifica dell'esempio che segue è un frammento di un progetto che verrà spiegato più avanti in questo capitolo; i puntini di sospensione rappresentano istruzioni che sono state volutamente tralasciate.

```
Dim hKey As Long, KeyIndex As Long

hKey = HKEY_LOCAL_MACHINE
KeyIndex = 0

Do While RegEnumIndex <> ERROR_NO_MORE_ITEMS
   RegEnumIndex = RegEnumKey(hKey, KeyIndex, szBuffer, lBuffSize)

Loop
```

Vediamo la dichiarazione della funzione RegEnumKey:

```
Declare Function RegEnumKey Lib "advapi32.dll" Alias _
   "RegEnumKeyA" (ByVal hKey As Long, _
   ByVal dwIndex As Long, ByVal lpName As String, _
   ByVal cbName As Long) As Long
```

Il terzo e il quarto parametro, ByVal lpName As String e ByVal cbName As Long, sono dichiarati rispettivamente di tipo stringa e lungo. In base alla documentazione di SDK, lpName contiene l'indirizzo di memoria del buffer relativo al nome della sottochiave e cbName rappresenta la dimensione del buffer (si veda il Capitolo 11). Comunque, se lavorate in Visual Basic come vi ho mostrato, la traduzione dalle API a VB funziona perfettamente.

Le istruzioni del registro incorporate in Visual Basic

Visual Basic 6 comprende quattro istruzioni incorForate per l'elaborazione del registro. Se questo quartetto è in grado di rispondere alle vostre esigenze, lo si può utilizzare facilmente, come vedremo tra un istante, senza storie, senza pasticci, senza dichiarazioni, senza niente da aggiungere.

Uno degli aspetti più ingegnosi di queste istruzioni riguarda il fatto che lavorano bene con sistemi operativi a 16 bit e a 32 bit. In Windows 95/98 sono in grado di leggere e scrivere nel registro di configurazione; in Windows 3-x queste stesse istruzioni leggono e scrivono in Win.ini. Se si sta lavorando sul codice di un'applicazione che deve lavorare in entrambi gli ambienti, questo è sicuramente un bel vantaggio!

Le istruzioni di impostazione delle applicazioni VB aggiungono e cancellano voci e valori del registro in HKEY_CURRENT_USER\Software. Queste istruzioni si aspet-

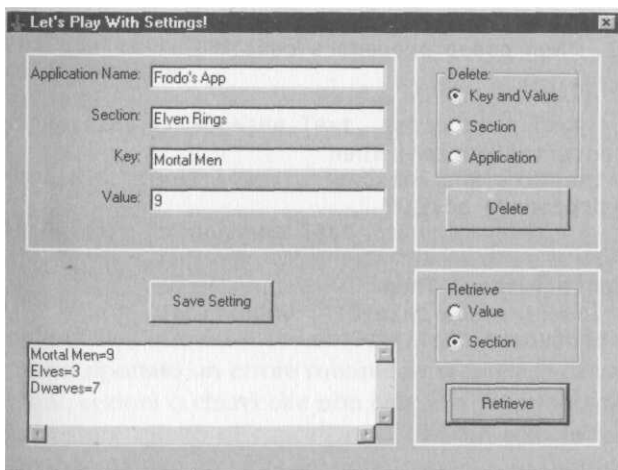
tano logicamente di leggere e scrivere applicazioni costituite da sezioni che contengono voci e valori; in altre parole, che siano file .Ini virtuali da sistemare all'interno del registro. Vediamone una rappresentazione schematica:

```
Mia Applicazione
   [Nome di sezione # 1 ]
      Chiave1=Valore
      Chiave2=Valore
   .......
   [Nome di sezione # 2]
  ........
```

Le istruzioni VB che manipolano questi file .Ini virtuali sono le seguenti:

- **DeleteSetting** elimina una chiave e il valore associato da un'applicazione e sezione particolari. Si può utilizzare DeleteSetting anche per cancellare un'intera sezione, se non viene definita alcuna chiave come parametro, e un'intera applicazione se non sono inclusi né sezione né parametri.
- **GetSetting** recupera un singolo valore dalla chiave richiamata nell'applicazione e sezione indicate.
- **GetAllSettings** riprende tutte le voci e i valori di una sezione.
- **SaveSetting** memorizza un valore della voce richiamata (nell'applicazione e sezione indicate).

Per illustrare la velocità e l'efficienza delle istruzioni incorForate che riguardano il registro, ho scritto una piccola applicazione che dimostra tutti i possibili utilizzi delle quattro istruzioni (si veda la Figura 10.1). Questo progetto è disponibile nel CD-ROM allegato al libro, nella directory relativa ai programmi del Capitolo 10; è stato salvato con il nome Settings. Vbp.

Figura 10.1 *Sipossono utilizzare facilmente le istruzioni incorporate VB del registro per memorizzare ericercare informazioni di inizializzazione (come in questa applicazione esempio).*

Vediamo come si utilizza l'istruzione SaveSetting (il codice si trova nella procedura dell'evento cmdSave_Click):

```
SaveSetting txtAppName.Text, txtSection.Text,
   txtKey.Text, txtValue.Text
```

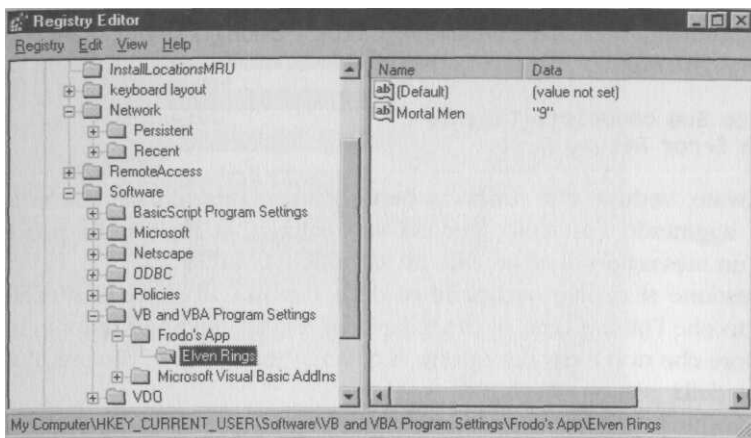
Veramente semplice, anche tenendo conto che SaveSetting non funziona se non si passano i nomi di un'applicazione, di una sezione e di una voce. (In effetti, il valore della voce non ha imFortanza: se txtValue.Text è vuoto, il valore aggiunto nel registro è 0.) Ho previsto una codifica che esegue una semplice verifica della validità dei campi di input; in altre parole, controlla se l'utente ha inserito una cosa qualsiasi.
Passare come parametro a una routine un controllo si rivela una tecnica utile quando sono richieste diverse *azioni* complesse al controllo che si deve passare.

```
Private Function TestContents(c As Control) As Boolean
   If c.Text = "" Then
      TestContents = False
   Else
      TestContents = True
   End If
End Function
```

Vediamo il resto del codice necessario per controllare la caselle di testo di input e per restituire i messaggi corrispondenti. (Ho utilizzato essenzialmente la stessa codifica che controlla che ci sia qualcosa nelle caselle di input nei diversi punti del programma. Avrei potuto semplificare il tutto spostando il codice in una subroutine ma, dato che ci sono alcune differenze nei diversi input da controllare, non ho voluto occuparmi di questo per un programma così semplice.)

```
If Not TestContents(txtAppName) Then
   MsgBox "Devi inserire il nome di una applicazione!",
      vbCritical, "Non posso procedere così!"
   Exit Sub
End If

If Not TestContents(txtSection) Then
   MsgBox "Devi inserire una sezione!", vbCritical,
      "Non posso procedere così!"
   Exit Sub
End If
If Not TestContents(txtKey) Then
   MsgBox "Devi inserire una chiave!", vbCritical,
      "Non posso procedere così!"
   Exit Sub
End If
```

Se si fa clic sul pulsante *SaveSetting* e si esegue la procedura, è possibile verificare mediante Regedit che sono stati aggiunti un'applicazione, una sezione, una chiave e un valore (si veda la Figura 10.2).

Figura 10.2 *Se si esegue Regedit si possono osservare le informazioni che sono state aggiunte nel registro utilizzando le istruzioni predefinite di Visual Basic.*

L'istruzione DeleteSetting può assumere tre diversi significati in una istruzione. In funzione del numero di parametri che vengono passati, è possibile cancellare una sola impostazione (chiave e valore), una intera sezione oppure tutte le impostazioni relative a un'applicazione. Vediamo il codice che implementa le tre modalità di DeleteSetting (ho tralasciato la parte che controlla l'input):

```
Private Sub cmdDelete_Click()
   If  optDelete(0).Value  Then
      'Cancella chiave e valore

      DeleteSetting txtAppName.Text, txtSection.Text, txtKey.Text
   ElseIf optDelete(l) Then
      'Cancella sezione

      DeleteSetting txtAppName.Text, txtSection.Text
   Else
      'Cancella tutte le impostazioni dell'applicazione

      DeleteSetting  txtAppName.Text
   End If
End Sub
```

Uno dei problemi dell'istruzione DeleteSetting (anche di GetSetting a dire il vero) è che viene riFortato un errore runtime se si cerca di cancellare o di recuperare applicazioni, sezioni o chiavi che non esistono. Un modo per risolvere questo problema può essere quello di cancellare o rilevare solo le impostazioni che la vostra applicazione ha trattato nella sessione corrente; in questo modo siete sicuri che le impostazioni esistono. Le impostazioni potrebbero essere create nel caricamento di un modulo, utilizzate dal modulo e cancellate dall'evento di scaricamento del modulo.

Questa procedura comForta tuttavia parecchie limitazioni dell'utilità di queste istruzioni. Una strategia migliore consiste nel gestire l'errore che deriva dal tentativo di cancellare o rilevare un'applicazione, una sezione o una chiave che non esistono.

Una soluzione di comodo è aggiungere un 'istruzione che fa ignorare a VB gli errori all'inizio delle procedure cmdDelete_Click *e* cmdRetrieve_Click, *che potrebbero causare l'errore:*

```
Private Sub cmdDelete_Click()
    On Error Resumé Next
```

Se provate, vedrete che funziona bene. L'unico problema è che si ignora il problema aggirando l'ostacolo, per cui se qualcosa va male nella procedura non si rileva un messaggio di errore che ne identifichi la causa.
La questione si risolve occupandosi della risposta all'errore particolare provocato dal fatto che l'utente tenti di cancellare dal registro un'applicazione, una sezione o un valore che non esistono; questo si chiama "intercettare" l'errore. Il Capitolo 15 si occupa della gestione degli errori.
In primo luogo è necessario identificare il numero dell'errore; potrebbe non essere quello che ci si aspetta. Per fare questo si provoca l'errore e si legge il messaggio corrispondente. Se si utilizza DeleteSetting per provare a cancellare qualcosa che non esiste nel registro si provoca un errore numero 5, il quale indica una chiamata non valida di una procedura. (La procedura cmdRetrieve_Click, che vedremo tra poco, provoca l'errore numero 13, tipo non corretto, quando l'utente prova a rilevare qualcosa che non c'è.)

// modo più semplice per individuare gli errori in Visual Basic, ed il loro significato, consiste nell'utilizzare l'indice e leggere l'argomento "Trappable Errors" nella guida di MSDN.

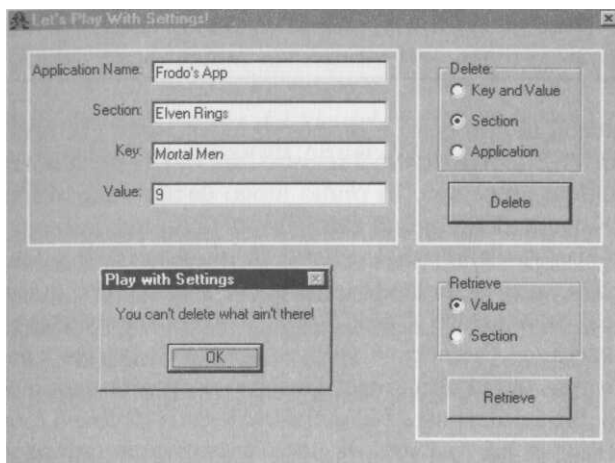
Il passo successivo consiste nell'aggiungere in testa alla procedura un salto al codice di gestione dell'errore. (Con un minimo di fantasia, si può assegnare al numero dell'errore una costante equivalente, che ne rende più chiaro l'utilizzo successivo.)

```
Private Sub cmdDelete_Click()
   Const ErrInvalidProcCall = 5
   On Error GoTo ErrHandle
```

Infine si aggiunge la gestione dell'errore in fondo alla procedura. Ci si deve assicurare di inserire un'istruzione Exit Sub prima dell'inizio del gestore dell'errore, in modo che non ci siano possibilità di "cadérci dentro" per sbaglio.

```
   Exit Sub
ErrHandle:
   If Err.Number = ErrInvalidProcCall Then
      MsgBox "You can't delete what ain't there!"
      Resumé Next
   End If
End Sub
```

Questo codice produce il messaggio di errore mostrato in Figura 10.3 nel caso in cui un utente cerca di cancellare dal registro qualcosa che non esiste.

Figura 10.3 *Sipossono utilizzare tecniche di individuazione degli errori che riFortino un messaggio appropriato nel caso in cui gli utenti cerchino di cancellare materiale che non esiste nel registro.*

A posto! Una volta sistemato questo, il Listato 10.1 mostra la parte finale della codifica del progetto che richiama la funzione GetSetting, la quale restituisce un valore, oppure la funzione GetAllSettings, che restituisce tutte le voci e i valori di una sezione. (Anche in questo caso ho tralasciato la codifica che verifica il contenuto effettivo nelle caselle di input.)

Listato 10.1 *Visualizzazione di valori individuati del registro.*

```
Private Sub cmdRetrieve_Click()
   Dim SectionSettings As Variant, IntX As Integer
   Const ErrTypeMismatch = 13
   On Error GoTo ErrHandle
   If optRetrieve(0).Value Then
      'Usa GetSetting per restituire un valore

      txtDisplay.Visible = True
      txtDisplay.Text = GetSetting(txtAppName.Text, _
         txtSection.Text, txtKey.Text, txtValue.Text)
     'textValue come impostazione di default nella sintassi
   Else 'Usa GetAllSettings per restituire tutta la sezione

     txtDisplay.Visible = True
     SectionSettings = GetAllSettings(txtAppName.Text, _
        txtSection.Text)
     Fon IntX = 0 To UBound(SectionSettings, 1)
        txtDisplay.Text = txtDisplay.Text & _
           SectionSettings(IntX, 0) & "=" &^ _
           SectionSettings(IntX, 1) & vbCrLf
     Next IntX
   End If
   Exit Sub
```

```
ErrHandle:
   If Err.Number = ErrTypeMismatch Then
      MsgBox "Non si può trovare quello che non c'è!"
      Resumé Next
   End If
End Sub
```

Siamo ora molto avanti nel progetto e sappiamo tutto quello che serve sulle istruzioni incorForate in VB che riguardano il registro, ma vale la pena notare un paio di aspetti di questa ultima procedura. In primo luogo GetSetting accetta un quarto parametro facoltativo, che definisce un valore predefinito nel caso in cui l'impostazione non esista o non sia definita nel registro; in questo caso il valore predefinito che viene passato alla funzione GetSetting è anche il valore che questa restituisce! In secondo luogo si può notare il modo con il quale SectionSettings è stato dichiarato come variante. Quando si utilizza SectionSettings come valore di ritorno per la funzione GetAllSettings, l'informazione riFortata a SectionSettings si trova nella forma di matrice bidimensionale (una di chiavi e una di valori). Si possono estrapolare le informazioni da SectionSettings trattandolo come una matrice (cioè per quello che è diventato). Mi è sembrato ingegnoso aggiungere un segno di uguaglianza tra ogni chiave e il valore corrispondente e inserire un'interruzione di riga tra le diverse voci, quando si legge la matrice in txtDisplay:

```
For IntX = 0 To UBound(SectionSettings, 1)
   txtDisplay.Text = txtDisplay.Text & _
      SectionSettings(IntX, 0) & "=" & _
      SectionSettings(IntX, 1) & vbCrLf
Next IntX
```

Utilizzo delle costanti VBA per la codifica dei comuni caratteri non stampabili

La libreria di costanti VBA comprende una serie di costanti predefinite che riguardano la codifica dei comuni caratteri non stampabili. Per esempio, vbCrLf in txtDisplay corrisponde a un ritorno carrello più nuova riga, che nelle vecchie versioni di VB equivaleva all'inserimento nella codifica di Chr$(13) + Chr$(10). Di seguito sono riFortate altre costanti relative a caratteri che è utile conoscere:

- Backspace (vbBack - Chr$(8))
- Carrìage return (vbCr = Chr$(13))
- Formfeed (vbFormFeed = Chr$(12))
- Line feed (vbLf = Chr$(10))
- Null (vbNullChar = Chr$(0))
- Tab (vbTab = Chr$(9))
- Vertical Tab (vbVerticalTab = Chr$(ll))

Si può facilmente utilizzare Object Browser per trovare tutte le costanti predefinite che fanno parte della libreria VBA.

Utilizzo delle API per manipolare il registro

le modifiche effettuate nel registro sono irreversibili, nel senso che hanno effetto immediato non appena si chiude il file del registro, senza ulteriori avvenimenti. La modifica di alcune impostazioni del registro può avere effetti disastrosi, come la disattivazione di alcune imFortanti funzioni del sistema. È quindi buona norma predisporre una copia di riserva del file del registro, selezionando la voce ExFort Registry File *dal menu* Edit *di Regedit, prima difare esperimenti con il registro della propria macchina.*

Le istruzioni del registro incorForate in VB funzionano molto bene per fare quello che sono destinate a fare (sono anche molto facili da usare) ma spesso capita di trovarsi in situazioni nelle quali occorre accedere direttamente al registro. Se si aggiunge il modulo RegAPI.Bas al proprio progetto, come descritto in precedenza in questo capitolo, e si utilizzano direttamente le API del registro, è possibile manipolare il registro senza le limitazioni introdotte dalle istruzioni incorForate in Visual Basic.
È facile ricercare chiavi, sottochiavi e valori nel registro; è possibile anche, senza troppi problemi, aggiungere nuove chiavi e valori oppure cancellare delle chiavi. L'elenco delle API pertinenti nella prima sezione di questo capitolo dà un'idea delle enormi possibilità a disposizione. Attenzione però, non sto certo dicendo di utilizzare RegUnloadKey per paralizzare tutto il software della concorrenza! Neanche per scherzo!

Ricerca e visualizzazione di chiavi e sottochiavi

Supponiamo di ricercare e visualizzare un ramo di chiavi all'interno di HKEY_LOCAL_MACHINE e tutte le sottochiavi di ogni chiave di livello superiore. L'idea di fondo consiste qui nel recuperare due livelli di chiavi, non l'intera struttura del registro.
Per visualizzare i due livelli di chiavi si può utilizzare un controllo TreeView, uno dei controlli personalizzati di Windows che sono stati discussi nel Capitolo 8; il risultato è simile a quello che si ottiene in Regedit, anche se ovviamente si possono utilizzare a piacere icone differenti. A me piace l'idea che il primo livello di chiavi venga rappresentato da un'icona a forma di sole, mentre il ramo interno da una a forma di terra.

Vediamo come impostare il controllo TreeView. *Si aggiungono un controllo* TreeView *e uno* ImageList *a un nuovo modulo (nel codice di esempio il modulo si chiama frmDisplay). Ci si deve assicurare di aggiungere alprogetto anche il modulo di dichiarazioni API. Il progetto si trova nel CD-ROM allegato al libro con il nome RegDisp. Vbp.*

Aggiungete il codice seguente alla procedura frmDisplay_Resize.-

```
Private Sub Form_Resize()
   TreeView1.Height = frmDisplay.ScaleHeight
   TreeView1.Width = frmDisplay.ScaleWidth
End Sub
```

Questo codice fa sì che il controllo TreeView abbia sempre le dimensioni stabilite dall'area client di frmDisplay; in questo modo la si può ridimensionare dato che cambia le dimensioni quando l'utente modifica quelle di frmDisplay.
Successivamente si utilizza la proprietà Custom di ImageList1 nella finestra *Properties* per aggiungere le due icone (il sole e la terra) nel controllo ImageList (si veda la Figura 10.4).

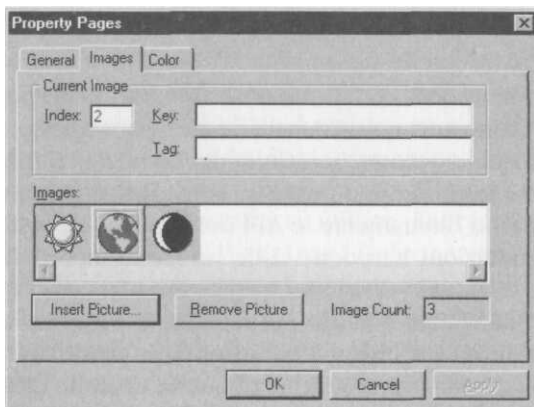
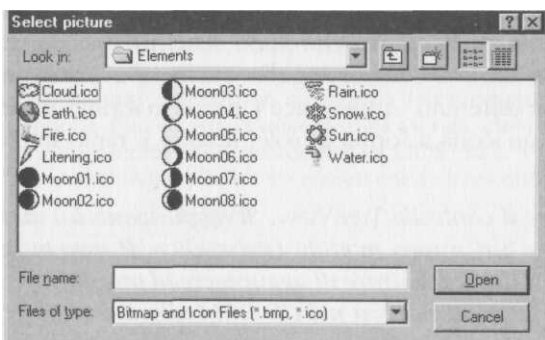
Figura 10.4 *Si può utilizzare la scheda Images del dialogo Property Pages di ImageList per aggiungere immagini alla libreria visuale memorizzata nel controtto.*

Le immagini di sole, terra e luna che ho aggiunto alla libreria visuale del controlli *ImageList* in questo progetto sono icone che ho prelevato da Visual Studio 6; si trovano nella directory Elements sotto Common\Graphics\Icons.
Un aspetto molto brillante della shell di Windows è costituito dal fatto che il dialogo generico dei file visualizza l'aspetto reale dell'icona collegata a un file che, nel caso di un file .Ico che contenga un'icona, è tutto quello che serve (si veda la Figura 10.5)!

Figura 10.5 *Quando si aggiunge un'immagine, come un 'icona, epossibilevedere un 'anteprima del suo aspetto.*

A questo proposito, mentre è ancora aperto il dialogo *Property Pages* di ImageList, selezionate la scheda *Generale* definite la dimensione di 16X16 pixel; questa è la più piccola dimensione possibile e va bene per la visualizzazione nel controllo Tree-View.

Si deve definire la dimensione nel dialogo Property Pages *di* ImageList *prima di caricare le immagini in* ImageList. *Una volta che il controllo contiene le immagini, non è più possibile modificarne le dimensioni (ehi, della Microsoft, mi sentite?). In altre parole, una volta che le immagini sono state caricate nel controllo, la dimensione delle immagini è in sola lettura. Se avete commesso un errore di dimensionamento, dovete ricominciare dall'inizio.*

Torniamo alla visualizzazione delle voci di HKEY_LOCAL_MACHINE; inseriamo un paio di dichiarazioni nella sezione General Declarations di frmDisplay:

```
Option Explicit
'Variabili globali
Public hKey As Long
Public NextLevel As Long
```

A questo punto, inseriamo il codice relativo all'evento di caricamento di frmDisplay per definire il progetto:

```
Private Sub Form_Load()
   Dim KeyIndex As Long, RegEnumIndex As Long,
      szBuffer As String, lBuffSize As Long,
      Nodkey As Mode, phkResult As Long, Indent As Long
   'Crea un buffer per le chiamate API
   szBuffer = Space(255)
   lBuffSize=Len(szBuffer)
   hKey = HKEY_LOCAL_MACHINE
   Indent = tvwChild
   KeyIndex = 0
   NextLevel = 0

   TreeView1.ImageList= ImageList1
   'Collega TreeView1 e ImageList1
```

La proprietà ImageList del controllo TreeView viene utilizzata per collegare il controllo con il controllo ImageList (che serve da libreria di immagini). La variabile Indent registra lo stato attuale dei nodi che sono stati aggiunti all'ImageList; twwChild è il valore predefinito. Si tratta di una costante, definita equivalente a 4, e significa che il nodo aggiunto al controllo TreeView diventa figlio di quello corrispondente alla chiamata del metodo Add dell'insieme Nodes. Selezionare "Add Method (Nodes Collection)" nella guida online per esaminare un elenco di tutti i valori corrispondenti. Si può anche notare che la variabile Nodkey è stata dichiarata di tipo Node.

KeyIndex è una variabile contatore utilizzata in RegEnumKey per definire un ciclo su tutte le sottochiavi della chiave di riferimento (in questo caso, HKEY_LOCAL_MACHINE). NextLevel tiene traccia di dove ci troviamo come livello dell'albero quando si aggiungono le immagini al componente TreeView. (Si può utilizzare questa variabile se si vuole espandere il progetto di un altro livello, oppure in modo ricorsivo per scorrere tutti i nodi al di sotto del punto di partenza.)

Il Listato 10.2 mostra la parte finale della procedura.

Listato 10.2 *Visualizzazione dei nodi.*

```
Do While RegEnumIndex <> ERROR_NO_MORE_ITEMS
      RegEnumIndex = RegEnumKey(hKey, KeyIndex, szBuffer, IBuffSize)
      If RegEnumIndex <> ERROR_SUCCESS And _
         RegEnumIndex <> ERROR_NO_MORE_ITEMS Then
            MsgBox "Errore in lettura!"
         Exit Do
      End If
      If szBuffer <> Space(255) Then
         Set Nodkey = TreeView1.Nodes.Add(, Indent, , szBuffer, 1)
         RegOpenKeyEx hKey, szBuffer, 0, 1, phkResult
         NextLevel = NextLevel + 1
         DisplayKey phkResult, TreeView1.Nodes.Count, Indent
         NextLevel = NextLevel - 1
      End If
      szBuffer = Space(255)
      KeyIndex = KeyIndex + 1
   Loop
   Nodkey.EnsureVisible
   End Sub
```

ERROR_SUCCESS e ERROR_NO_MORE_ITEMS sono costanti dichiarate nel modulo RegAPI.Bas; costituiscono due fra i codici che possono essere restituiti dalla funzione RegEnumKey.
La logica di questa procedura prevede un ciclo che si ripete fino a quando RegEnumKey non restituisce ERROR_NO_MORE_ITEMS (il che significa, "non ci sono più chiavi a questo livello, amico"), incrementando di uno il contatore KeyIndex ad ogni passaggio. Non è prevista intercettazione dell'errore; se RegEnumKey restituisce un valore diverso da ERROR_SUCCESS ("ho trovato una chiave") e da ERROR_NO_MORE_ITEMS, il ciclo viene interrotto. Conviene fare questo tipo di controllo del codice di ritorno dell'API del registro per essere sicuri che la funzione si comForti correttamente.
Se la funzione è andata a buon fine, si controlla il contenuto di szBuffer. Se è presente qualcosa, si aggiunge un nodo di primo livello al controllo TreeView utilizzando il contenuto di stringa di szBuffer. Il parametro finale del metodo Add, 1, indica il valore indice dell'immagine del controllo ImageList che rappresenta il nodo. (Si può aggiungere un altro parametro costituito da un numero indice di ImageList che rappresenti il nodo quando viene selezionato.)
Poi si utilizza la funzione RegOpenKeyEx per riFortare il gestore phkResult relativo alla sottochiave chiamata in szBuffer. Grazie a questo dato viene chiamata DisplayKey, una routine che restituisce il livello successivo di chiavi e le sistema nel controllo TreeView.

Il Listato 10.3 mostra il contenuto di DisplayKey.

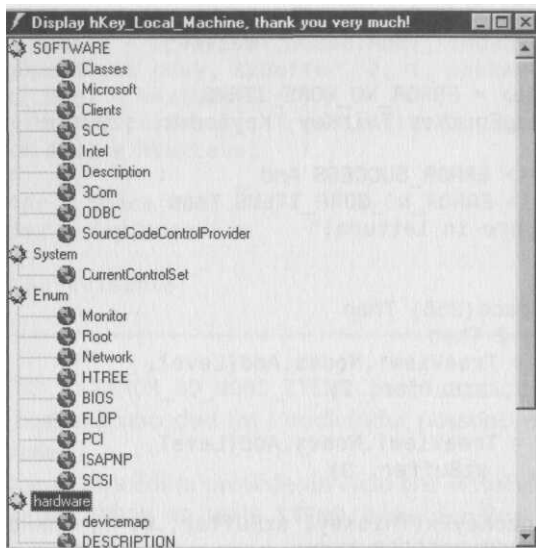
Listato 10.3 *Visualizzazione del contenuto di una chiave.*

```
Public Sub DisplayKey(ThisKey As Long, Level As Long, _
    Indent As Long)
    Dim KeyIndex As Long, RegEnumIndex As Long, _
        szBuffer As String, lBuffSize As Long,
        Nodkey As Node, phkResult As Long, RetKey As Long

    szBuffer = Space(255)
    lBuffSize = Len(szBuffer)
    KeyIndex = 0
    Indent = tvwChild

    Do Until RegEnumIndex = ERROR_NO_MORE_ITEMS
        RegEnumIndex = RegEnumKey(ThisKey, KeyIndex, szBuffer, _
            lBuffSize)
        If RegEnumIndex <> ERROR_SUCCESS And _
            RegEnumIndex <> ERROR_NO_MORE_ITEMS Then
                MsgBox "Errore in lettura!"
                ExitDo
        End If
        If szBuffer <> Space(255) Then
            If NextLevel < 2 Then
                Set Nodkey = TreeView1.Nodes.Add(Level, _
                    Indent, , szBuffer, 2)
            Else
                Set Nodkey = TreeView1.Nodes.Add(Level, _
                    Indent, , szBuffer, 3)
            End If
            RetKey = RegOpenKeyEx(ThisKey, szBuffer, 0, 1, phkResult)
            If RetKey = ERROR_SUCCESS Then
                Indent = tvwChild
                NextLevel = NextLevel + 1
                'Qui si effettua una chiamata ricorsiva
                'per visualizzare il livello suvvessivo!
                'DisplayKey phkResult, NextLevel, Indent
                NextLevel = NextLevel - 1
            End If
        End If
        szBuffer = Space(255)
        KeyIndex = KeyIndex + 1
    Loop
End Sub
```

Si può notare che il codice prevede la visualizzazione di una terza icona, una luna, relativa al livello successivo in basso, nel caso in cui il codice venisse modificato per arrivarci. Questo frammento è stato pensato in modo da poter essere modificato facilmente per tenere traccia in modo ricorsivo dell'intero ramo HKEY_LOCAL_-MACHINE del registro.

Nel caso voleste modificare il programma perché lavori in modo ricorsivo, devo avvisarvi: tenete a Fortata di mano qualcosa da fare, quando lo lanciate. Ci vorrà un po' di tempo prima che finisca.

Se si esegue il programma, si ottiene un ramo del registro proprio bello da vedere, con icone poco consuete (si veda la Figura 10.6). Questo progetto potrà anche non essere utile in quanto tale, ma nel caso in cui fosse necessaria un'analisi di registri particolari, si possono utilizzare queste tecniche per avere velocemente una mappa di quello che serve.

Figura 10.6
Si può utilizzare l'API del registro per accedere e visualizzare il contenuto del registro; in questo caso si vedono i due livelli superiori del ramo HKEY_LOCAL_-MACHINE.

Ricerca e modifica di valori

Spesso si ha la necessità di recuperare il valore corrispondente a una determinata chiave. A volte si vuole modificare questo valore e memorizzarlo di nuovo nel registro. A patto di conoscere il nome della chiave che contiene il valore e la sua posizione nella struttura del registro, questo compito è abbastanza facile da svolgere grazie alle funzioni RegOpenKeyEx, RegQueryValueEx e RegSetValueEx.

Vi sto per mostrare come recuperare un valore relativo al progetto di esempio che legge i nomi predefmiti dell'utente e della società contenuti nelle caselle di testo di un modulo (memorizzato nel CD-ROM allegato al libro con il nome Values. Vbp). Questo progetto funziona in modo simile a molte routine di inizializzazione: si chiede all'utente di modificare oppure confermare il nome e la società predefiniti durante l'installazione di nuovo software. Una funzione del programma dimostrativo che di solito non è presente nei programmi di installazione riguarda il fatto che l'utente memorizza nel registro le nuove informazioni quando fa clic sul pulsante Apply.

Una parola, e alcune avvenenze, che riguardano la posizione del registro. Nel programma esempio le voci relative alle informazioni predefinite dell'utente sono le chiavi utilizzate da Microsoft e da altri per ricavare le informazioni durante le routine di installazione (che vedremo più avanti in questo capitolo). Tuttavia, queste informazioni sono memorizzate in molti posti diversi del registro (e diventa tutto ancora più complicato quando una macchina è configurata per più utenti).

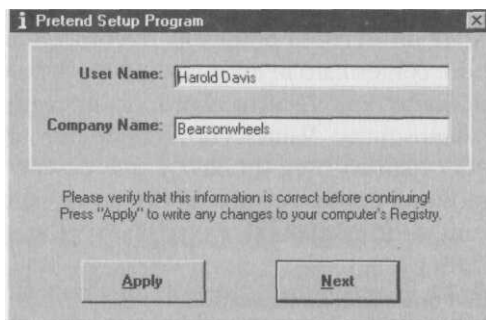
Per dirla in altre parole, le applicazioni che creo io possono (e di solito lo fanno) memorizzare informazioni sull'utente in proprie chiavi. A meno di utilizzare Regedit per spulciare tutto il registro, non c'è modo di stabilire la posizione o il nome di quelle chiavi. I principali fornitori di software come Microsoft sono capaci di memorizzare informazioni sull'utente relative a differenti applicazioni in posti diversi fra loro. La funzione GetUserName, discussa più avanti in questo capitolo, costituisce un modo per ottenere informazioni sul nome dell'utente più semplice del metodo utilizzato nell'esempio. Il programma esempio segue l'orientamento di molti programmi di installazione di Microsoft e ricava informazioni sul nome predefinito dell'utente dalla chiave *DefName* di HKEY_USERS\.Default\Software\Microsoft\MS Setup (ACME)\User Info. Attenzione al punto che precede Default: è necessario! L'API GetUserName, invece, utilizza la chiave *Current User* di HKEY_LOCAL_MACHINE\System\Current\ControlSet\Control.
La scelta dipende dalle proprie preferenze. È ovvio che si può impostare RegQueryValueEx in modo che legga dalla locazione GetUserName, se lo si desidera. Il punto cruciale è che non si deve mai presupForre che l'informazione predefinita nel registro sia corretta senza dare all'utente la possibilità di correggerla. Una volta che l'utente ha modificato oppure confermato le informazioni predefinite, conviene memorizzarle per conto proprio in un ramo relativo al proprio software.
Per impostare il programma dimostrativo Values.Vbp, aggiungete al progetto il modulo di dichiarazione API del registro (RegAPI.Bas) e due caselle di testo nel form di avvio. Il Listato 10.4 mostra il codice relativo all'evento di caricamento del form che sistema i corrispondenti valori predefiniti:

Listato 10.4 *Sistemazione dei valori predefiniti in un modulo.*

```
Private Sub Form_Load()
   Dim szBuffer As String, dataBuff As String, _
      IdataBuffSize As Long, hKey As Long, phkResult As Long, _
      RetVal As Long, Value As String, RegEnumIndex As Long

   'Crea Buffer
   dataBuff = Space(255)
   IdataBuffSize = Len(dataBuff)

   szBuffer = " .Default\Software\Microsoft\MS Setup (ACME)\User  Info"
   hKey = HKEY_USERS
   RetVal = RegOpenKeyEx(hKey, szBuffer, 0, KEY_ALL_ACCESS, _
      phkResult)
   If RetVal = ERROR_SUCCESS Then MsgBox "OKDokey"

   Value = "DefCompany"
   RetVal = RegQueryValueEx(phkResult, Value, 0, 0, dataBuff, _
```

```
      IdataBuffSize)
   If RetVal = ERROR_SUCCESS Then
      'Elimina il terminatore nullo e legge nella casella di testo
      txtCompany.Text = Left(dataBuff, IdataBuffSize - 1)
   Else
      MsgBox "Errore interno in RegQueryValueEx"
   End If

   Value = "DefName"
   RetVal = RegQueryValueEx(phkResult, Value, 0, 0, dataBuff, _
      IdataBuffSize)
   If RetVal = ERROR_SUCCESS Then
      txtName.Text = Left(dataBuff, IdataBuffSize - 1)
   Else
      MsgBox "Mancato RegQueryValueEx al secondo passaggio!"
   End If

         'Chiude le chiavi
   RegCloseKey hKey
   RegCloseKey phkResult
End Sub
```

Se si esegue il programma, si può notare che questo codice non fa altro che Fortare alla luce il nome dell'utente e dell'organizzazione predefiniti (si veda la Figura 10.7), o meglio quelli definiti dai valori delle apposite chiavi.

Figura 10.7 *Si può utilizzare l'API del registro per recuperare l'utente e il nome della società predefiniti.*

RegQueryValueEx *può a volte riservare sorprese nei valori che restituisce. È imFortante verificare il ritorno corretto dellafunzione (verìfica svolta nel codice d'esempio grazie alla costante* ERROR_SUCCESS>), *prima di fare qualunque cosa con il valore che si ottiene. Inoltre, ho notato chefunziona meglio se si chiamano le chiavi e i valori desiderati secondo l'ordine con il quale compaiono nella struttura del registro.*

È utile anche verificare che la funzione RegQueryValueEx funzioni correttamente durante il processo di debug. In primo luogo, questo serve a garantire che il percorso del registro venga inserito correttamente; qualunque discrepanza può provocare un errore. La codifica da me utilizzata per controllare la funzione è indicata come commento nell'esempio:

```
'If RetVal = ERROR_SUCCESS Then MsgBox "OKDokey"
```

Il Listato 10.5 mostra il codice che trasferisce le modifiche nel registro.

Listato 10.5 *Nuovo inserimento di modifiche nel registro.*

```
private Sub cmdApply_Click()
  Dim NewName As String, NewCompany As String, phkResult As Long, _
     sSetValue As String, sValue As String, hKey As Long,_
     szBuffer As String, RetVal As Long

     NewName = txtName.Text
  NewCompany = txtCompany.Text
  szBuffer = ".Default\Software\Microsoft\MS Setup (ACME)\User Info'
  hKey = HKEY_USERS

     RetVal = RegOpenKeyEx(hKey, szBuffer, 0, KEY_ALL_ACCESS, _
     phkResult)
  'If RetVal = ERROR_SUCCESS Then MsgBox "OKDokey"
  sSetValue = "DefCompany"
  sValue = NewCompany
  RetVal = RegSetValueEx(phkResult, sSetValue, 0, REG_SZ, sValue, _
     CLng(Len(sValue) + 1 ))
  If RetVal <> ERROR_SUCCESS Then _
     MsgBox "Impossibile scrivere nel Registro!"

      sSetValue = "DefName"
  sValue = NewName
  RetVal = RegSetValueEx(phkResult, sSetValue, 0, REG_SZ, sValue, _
     CLng(Len(sValue) +1 ) )
  If RetVal <> ERROR_SUCCESS Then _
     MsgBox "Impossibile scrivere nel Registro!"
  'Close the keys
  RegCloseKey hKey
  RegCloseKey phkResult
End Sub
```

Grazie al codice scritto per il progetto d'esempio, è possibile modificare il nome dell'utente e della società predefiniti nel registro. Se si definiscono il nome "Katherine Janeway" e la società "Federation Starship Voyager", è possibile eseguire Regedit.Exe per esaminare le chiavi e i valori corretti e per verificare che siano stati inseriti al posto giusto (si veda la Figura 10.8).

Per essere sicuri che anche qualcun altro si preoccupa di leggere il registro e il particolare insieme di chiavi e valori che sono utilizzati da questo programma dimostrativo, si può constatare che le modifiche apFortate utilizzando Values. Vbp vengono rilevate dal programma di installazione della Library Edition del CD-ROM di Microsoft Developer Network (si veda la Figura 10.9).

Figura 10.8 *Si possono scrivere nuovi valori predefiniti nel registro utilizzando l'API RegSetValueEx.*

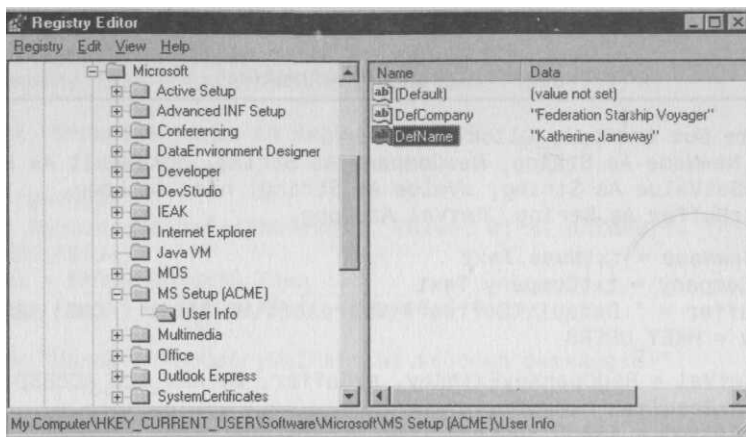
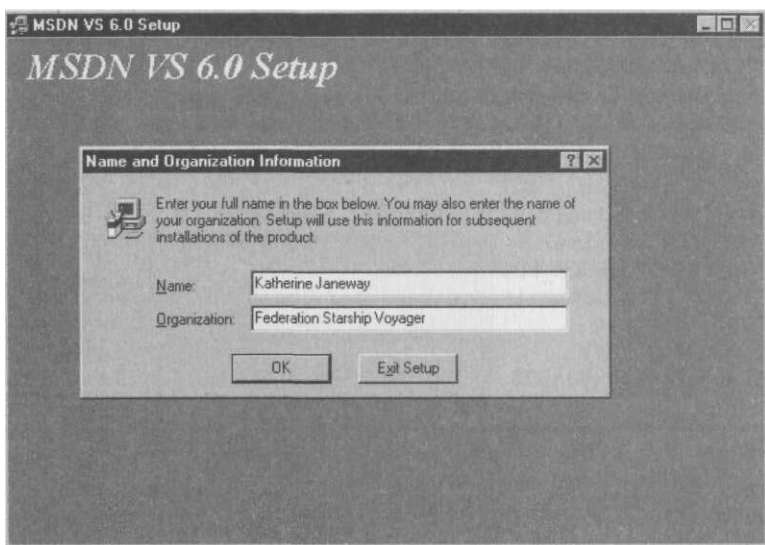



Figura 10.9 *Il programma di installazione per Visual Studio Edition di Microsoft Developer Network rileva le informazioni modificate dell'utente.*

In definitiva, la lettura e la scrittura di valori nel registro da parte dei programmi consentono di aggiungere un tocco professionale al vostro lavoro.

Eliminazione e inserimento di un terminatore nullo

È buona norma liberarsi del terminatore nullo quando si riFortano le stringhe in stile C in Visual Basic, altrimenti il terminatore nullo Chr(O) può comparire nella visualizzazione del testo come un blocco nero (letteralmente). Per liberarsi del terminatore nullo, si ritaglia l'ultimo carattere utilizzando la funzione Left una volta che è stata restituita la stringa:

```
txtCompany.Text = Left(dataBuff, IdataBuffSize - 1)
```

In modo analogo, quando si riFortano indietro le stringhe occorre aggiungere un carattere alla variabile lunghezza per memorizzare il terminatore nullo. Si utilizza oer questo il parametro finale nella chiamata della funzione API, che indica la lunghezza della stringa. Si può utilizzare facilmente la funzione CStr per restituire un carattere in più (il carattere nullo) del numero di caratteri nel buffer:

```
RetVal = RegSetValueEx(phkResult, sSetValue, 0, REG_SZ, sValue,
   CLng(Len(sValue) + 1))
```

NT a confronto di 95/98

Le stringhe di Windows 95/98 che sono valori chiave nel registro hanno sempre un terminatore nullo, mentre i valori chiave di NT 4.0 non vengono sempre gestiti in questo modo. Vediamo una funzione del modulo RegAPI.Bas che cancella un terminatore nullo, se esiste, altrimenti riForta la stringa intatta:

```
Public Function ConvertString(tmpVal As String, _
   KeyValSize As Long) As String
   If (Asc(Mid(tmpVal, KeyValSize, 1)) = 0) Then
      'Stringa Win95, elimina il terminatore nullo
      ConvertString = Left(tmpVal, KeyValSize - 1)
   Else
      'WinNT non ha un carattere nullo alla fine delle stringhe
      ConvertString = Left(tmpVal, KeyValSize)
      'Non trovato Null, restituisce solo la stringa
   End If
End Function
```

Nel CD-ROM allegato al libro è presente il modulo Convert.Bas che contiene una funzione GetKeyValue, *la quale incapsula ilprocedimento di conversione e restituzione dei valori chiave.*

GetUserName

Un modo veloceper rilevare il nome dell'utente corrente consiste nell'utilizzare l'API GetUserName. *Il Listato 10.6 mostra la codifica che dichiara l'API e la utilizza per trovare il nome dell'utente, posto in unprogetto che nonprevede moduli diform.*

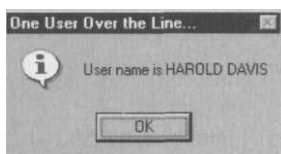
Ilprogetto API GetUserName èpresente nel CD-ROM allegato al libro con il nome di UserName. Vbp.

Listato 10.6 *Utilizzo di GetUserName.*

```
Option Explicit
Declare Function GetUserName Lib "advapi32.dll" Alias _
   "GetUserNameA" (ByVal IpBuffer As String, _
   nSize As Long) As Long

Public Sub Main()
   Dim szBuffer As String, IBuffSize As Long, RetVal As Boolean
   'Crea Buffer
   szBuffer = Space(255)
```

```
    lBuffSize = Len(szBuffer)
    RetVal = GetUserName(szBuffer, lBuffSize)
    If RetVal Then
       MsgBox "User name is " & UCase(Trim(szBuffer)), _
          vbInformation, "One User Over the Line..."
    Else
       MsgBox "GetUserName failed...sob!", vbInformation, _
          "One User Over the Line..."
    End If
End Sub
```

Per impostare il progetto, avviatene uno nuovo con l'opzione di avvio impostata a *SubMain.* Aggiungete un modulo .Bas al progetto, inserire una procedura di nome Main. Aggiungere al modulo il codice appena visto. Togliete il form vuoto dal progetto. Se si esegue il progetto, compare un messaggio con il nome dell'utente (si veda la Figura 10.10).

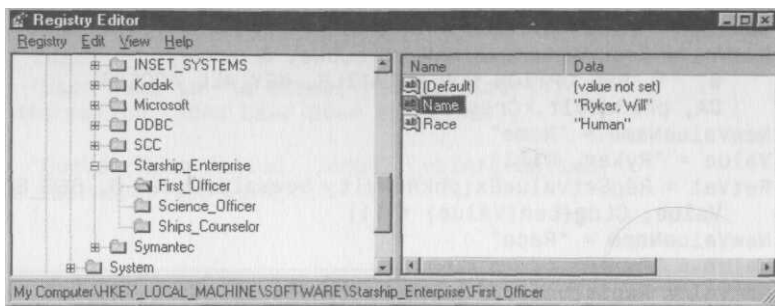
Figura 10.10 *L'API GetUserName riporta il nome corrente dell'utente.*

Inserimento ed eliminazione di chiavi

Le funzioni RegCreateKeyEx *e RegSetValueEx facilitano il compito di inserimento di chiavi e il successivo inserimento di nuovi valori nelle chiavi. Il Listato 10.7 contiene la codifica che aggiunge una chiave a Starship Enterprise, una serie di chiavi interne relative agli ufficiali chiave (perdonate il gioco di parole) oltre a nuovi valori relativi alla chiave di ciascun ufficiale (nome e razza). La Figura 10.11 mostra la struttura risultante come viene visualizzata da Regedit. Il progetto è presente nel CD-ROM allegato al libro con il nome di Keys. Vbp.*

Listato 10.7 *Inserimento di chiavi e valori nel registro.*

```
Public Sub Main()
    Dim RetVal As Long, hKey As Long, subkey As String, _
        newkey As String, phkResult As Long, _
        SA As SECURITY_ATTRIBUTES, Create As Long, _
        NewValueName As String, Value As String
    hKey = HKEY_LOCAL_MACHINE
    subkey = "SOFTWARE\"
    newkey = "Starship_Enterprise"
    RetVal = RegCreateKeyEx(hKey, subkey & newkey, _
        0, "", REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, _
        SA, phkResult, Create)
```

```
    subkey = AddASlash(subkey & newkey)
    newkey = "First_Officer"
    RetVal = RegCreateKeyEx(hKey, subkey & newkey,
        0, "", REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, _
        SA, phkResult, Create)
    NewValueName = "Name"
    Value = "Ryker, Will"
    RetVal = RegSetValueEx(phkResult, NewValueName, 0, REG_SZ, _
        Value, CLng(Len(Value) +1))
    NewValueName = "Race"
    Value = "Human"
    RetVal = RegSetValueEx(phkResult, NewValueName, 0, REG_SZ, _
        Value, CLng(Len(Value) + 1))
    'Chiude la chiave
    RegCloseKey phkResult

      newkey = "Ships_Counselor"
    RetVal = RegCreateKeyEx(hKey, subkey & newkey,
        0, "", REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, _
        SA, phkResult, Create)
    NewValueName = "Name"
    Value = "Troy, Deana"
    RetVal = RegSetValueEx(phkResult, NewValueName, 0, REG_SZ, _
        Value, CLng(Len(Value) +1))
    NewValueName = "Race"
    Value = "Betazoid"
    RetVal = RegSetValueEx(phkResult, NewValueName, 0, REG_SZ, _
        Value, CLng(Len(Value) +1))
    'Chiude la chiave
    RegCloseKey phkResult

     newkey = "Science_Officer"
    RetVal = RegCreateKeyEx(hKey, subkey & newkey,
        0, "", REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, _
        SA, phkResult, Create)
    NewValueName = "Name"
    Value = "DATA"
    RetVal = RegSetValueEx(phkResult, NewValueName, 0, REG_SZ, _
        Value, CLng(Len(Value) +1))
    NewValueName = "Race"
    Value = "È un androide, stupido!"
    RetVal = RegSetValueEx(phkResult, NewValueName, 0, REG_SZ, _
        Value, CLng(Len(Value) +1))
    'Chiude la chiave
  RegCloseKey phkResult
End Sub
```

Figura 10.11 *Si possono utilizzare le API del registro per creare voci gerarchiche; per esempio, ho inserito nel mio registro la Starship Enterprise!*

Ogni volta che in questo codice si vuole creare il percorso a una nuova sottochiave, si richiama la funzione AddASlash. *Per esempio:*

```
subkey = AddASlash(subkey & newkey)
```

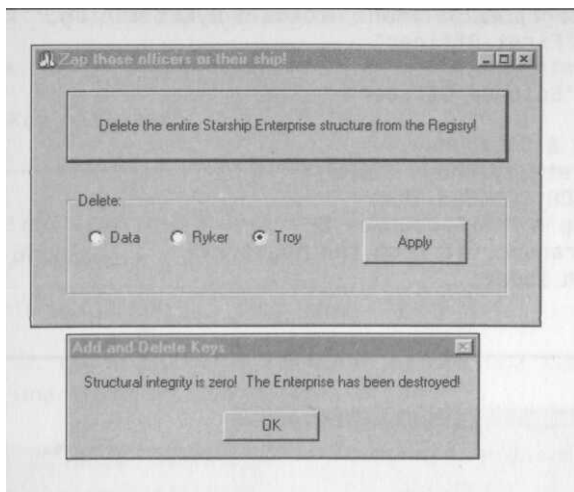
AddASlash aggiunge semplicemente una barra rovesciata (\) alla fine della stringa che viene passata, se non ne esiste già una. Questo fatto è imFortante, dato che le sottochiavi del registro devono essere richiamate con barre rovesciate per separare i percorsi interni concatenati. La situazione nella codifica dell'esempio è particolarmente semplice, ma le cose potrebbero risultare molto più nidificate, nel qual caso AddASlash può essere veramente d'aiuto. Il Listato 10.8 mostra la funzione AddASlash.

Listato 10.8 *Inserimento di una barra rovesciata atta fine di un percorso.*

```
Public Function AddASlash(InString As String) As String
    'Aggiunge una "/" alla fine di InString, se non c'è.
    '  Così poi la concatenazione di sottochiavi è più facile.
    If Mid(InString, Len(InString), 1) <> "\" Then
        AddASlash = InString & "\"
    Else
        AddASlash = InString
    End If
End Function
```

Il prossimo compito riguarda l'eliminazione delle chiavi e dei valori che ho appena inserito nel registro. Per fare questo si utilizza la funzione RegDeleteKey.
ATTENZIONE: si deve procedere con cautela quando si utilizza RegDeleteKey, dato che questa cancella una chiave, tutte le sue sottochiavi e tutti i valori delle chiavi e delle sottochiavi.
In primo luogo si inserisce una chiamata del metodo Show di frmDelete alla fine della procedura Sub Main che crea la struttura Starship Enterprise:

```
frmDelete.Show
End Sub
```

Figura 10.12
Si può utilizzare RegDeleteKey per cancellare intere strutture del registro. Chi è stato a fare Questo? Un Borg? Forse i Klingon o i Romulani?

frmDelete consente all'utente di cancellare l'intera struttura Enterprise oppure solo quella relativa a una sotto voce (si veda la Figura 10.12).
Il Listato 10.9 contiene il codice che cancella l'intero ramo di Starship Enterprise.

Listato 10.9 *Eliminazione di un ramo del registro.*

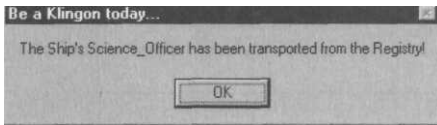
```
Private Sub cmdDeleteShip_Click()
   Dim RetVal As Long, hKey As Long, szBuffer As String
   hKey = HKEY_LOCAL_MACHINE
   szBuffer = "SOFTWARE\Starship_Enterprise"
   RetVal = RegDeleteKey(hKey, szBuffer)
   If RetVal = ERROR_SUCCESS Then
      MsgBox "Structural integrity is zero!" & _
         "The Enterprise has been destroyed!"
   End If
End Sub
```

Se si vuole cancellare una singola chiave e tutti i suoi valori, si procede in modo analogo, come mostrato nel Listato 10.10. Il messaggio di Figura 10.13 indica che è stato restituito il flag ERROR_SUCCESS, e quindi l'operazione ha avuto successo.

Listato 10.10 *Eliminazione di una chiave e dei suoi valori.*

```
Private Sub cmdApply_Click()
   Dim RetVal As Long, hKey As Long, SubKey As String, _
      SubsubKey As String
   hKey = HKEY_LOCAL_MACHINE
   SubKey = AddASlash("SOFTWARE\Starship_Enterprise")

      If optDelete(2).Value Then 'zap Troy
      SubsubKey = "Ships_Counselor"
```

```
   ElseIf optDelete(1).Value Then ' tocca a Ryker morire
      SubsubKey = "First_Officer"
   Else 'By, by, Data
      SubsubKey = "Science_Officer"
   End If
   SubKey = SubKey & SubsubKey
   RetVal = RegDeleteKey(hKey, SubKey)
   If RetVal = ERROR_SUCCESS Then
   MsgBox "The Ship's " & SubsubKey & _
      " has been transForted from the Registry!", , _
      "Be a Klingon today..."
   End If
End Sub
```

Figura 10.13 *Questo messaggio arriva dai nostri amici e vicini di casa Klingon.*

Utilizzo di RegDeleteValue

Se si vuole solo cancellare il nome di un valore e il valore stesso (ma lasciare la chiave), si utilizza RegDeleteValue. Per esempio, si può conservare la chiave Starship_Enterprise\First_Officer, ma cancellare il nome del valore e il valore Race=Human.

RegDeleteValue funziona in modo analogo a RegDeleteKey tranne per il fatto che viene passato il nome del valore al posto del nome della sottochiave da cancellare. La codifica che consente di cancellare il nome del valore e il valore Race=Human si trova nel progetto esempio in corrispondenza dell'evento clic di frmDelete ed è mostrata nel Listato 10.11.

Listato 10.11 *Eliminazione di un valore.*

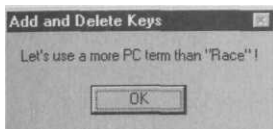
```
Private Sub Form_Click()
   Dim RetVal As Long, hKey As Long, ValueName As String, _
      SubKey As String, phkResult As Long, _
      SA As SECURITY_ATTRIBUTES, Create As Long
   hKey = HKEY_LOCAL_MACHINE
   SubKey = "SOFTWARE\Starship_Enterprise\First_Officer"
   RetVal = RegCreateKeyEx(hKey, SubKey,
      0, "", REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, _
      SA, phkResult, Create)
   ValueName = "Race"
   RetVal = RegDeleteValue(phkResult, ValueName)
   If RetVal = ERROR SUCCESS Then
```

```
    MsgBox "Let's use a more PC term than ""Race"" !"
  Else
    MsgBox "Error of some sort."
  End If
  RegCloseKey phkResult
End Sub
```

Nella routine qui sopra una delle chiamate MsgBox utilizza una stringa racchiusa tra virgolette doppie:

```
MsgBox "Let's use a more PC term than ""Race"" !"
```

È noto che in questo modo è possibile inserire una citazione fra virgolette all'interno di una stringa (si veda la Figura 10.14).

Figura 10.14 *Un messaggio che contiene virgolette doppie.*

Creazione di un componente ActiveX per incapsulare le API del registro

Soddisfacente! E ho proprio voglia di superarmi (sì, ancora una volta!): le applicazioni dei componenti ActiveX sono trattate nel Capitolo 23, mentre nel Capitolo 14 si possono avere informazioni sull'utilizzo dei moduli di classe.
Nonostante questo, la possibilità di creare facilmente applicazioni server dei componenti ActiveX costituisce uno degli aspetti più esaltanti di Visual Basic a 32 bit. Un componente ActiveX, il termine moderno per indicare un'applicazione server OLE, è una strada naturale che consente di raggruppare alcune delle chiamate di registro che sono state illustrate in questo capitolo. Una volta completato il componente ActiveX, quello che serve per accedere ai suoi metodi esposti consiste nell'assicurarsi che sia incluso tra i riferimenti del progetto e che sia presente una chiamata di funzione. È possibile utilizzarlo più volte senza la necessità di codificare nuovamente la logica della chiamata delle API del registro. Vita facile, davvero! Se si raggnippa tutto il codice in componenti ActiveX il lavoro risulta modulare, mantenibile e utilizzabile in qualunque punto e si può quindi avere più tempo per dedicarsi ad altro!

Questa sezione descrive passo dopo passo la creazione di un'applicazione server ActiveX che incapsula le API del registro utilizzate per rilevare e memorizzare i valori relativi alprogetto Values. Vbp. Dato che ci siamo già occupati in questo capitolo della logica che sta dietro il progetto, il lavoro non dovrebbe risultare difficile come sembra. Il progetto è presente nel CD-ROM allegato al libro con il nome di Server. Vbp. Una volta spiegato il server OLE, vedremo come creare un progetto di verifica che lo richiama (salvato su CD-ROM come TestSer. Vbp).

Creazione di un server ActiveX

Per creare il server ActiveX, seguite questi passaggi:

1. Nel dialogo *Project Options,* selezionate *Sub Main* come form di avvio, inserire un *Project Name* ("*ReadandWriteRegVals"}* e scrivete una descrizione dell'applicazione, come mostrato in Figura 10.15. Ci si deve assicurare che sia selezionato *ActiveX Component* come *Start Model* del progetto (si veda la Figura 10.16). La descrizione dell'applicazione compare nel dialogo *References* quando si arriva allo stadio di collaudo del server (si vedano il primo passo della prossima sezione e la Figura 10.18).

Figura 10.15
La definizione di Project Description in ProjectProperties viene utilizzata più avanti comeriferimento alserverActiveX.

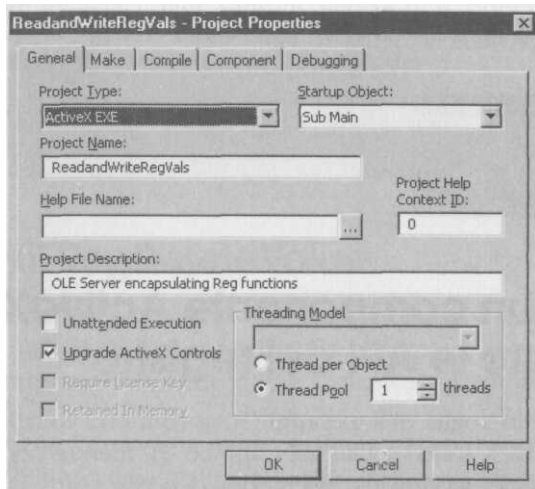
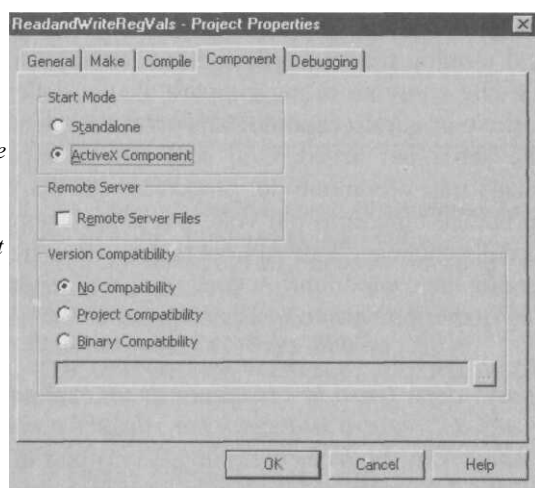
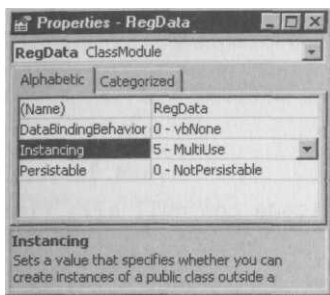



Figura 10.16
I serverActiveX devono essere definiti in modo da avviarsi come componente ActiveX nella scheda Component del dialogoProject Properties.

2. Eliminate dal progetto il *Form1* predefinito. I componenti ActiveX non hanno bisogno di form.

3 Aggiungete un modulo di codice al progetto. Aggiungete una procedura *SubMain* a questo file .Bas. Il file *SubMain* rimane senza codice. Assicuratevi che il progetto sia impostato per un avvio da *SubMain*.

4 Aggiungete un modulo di classe al progetto. Visualizzate la finestra *Properties* del modulo (si accede a questa finestra dal menu di scelta rapida del modulo della classe). Come nome *(Name)* del modulo della classe indicate RegData e impostate *Instancing* come *5-MultiUse* (si veda la Figura 10.17). Chiudete la finestra *Properties*.

Figura 10.17 *Si utilizzano le finestre Properties per definire le proprietà Instancing di un modulo* Class.

Inserite una nuova *Property Procedure* di nome *Value* nel modulo della classe. La proprietà deve essere Public come ambito d'azione. Al modulo della classe verrà aggiunto questo codice modello:

```
Public Property Get Value()

End Property

Public Property Let Value(vNewValue)

End Property
```

Si usano Get Property per prelevare i valori dal registro e Let per memorizzare i valori nel registro. Il modo di lavorare prevede che Property e Value possano essere utilizzati da entrambi i membri dell'equivalenza. Si richiama Property Get come se fosse una funzione normale. Ad ogni modo le liste dei parametri formali di Get e Let per una Property devono combaciare per nomi e tipi. Il nome vNewValue non deve essere modificato e coincide con quanto restituito dalla funzione Property Get. Se non si seguono queste regole si ottiene un errore di sintassi in runtime. Per esempio:

```
Public Property Get Value(hKey As Long, SubKey As String, _
   ValueName As String) As String

End Property

Public Property Let Value(hKey As Long, SubKey As String, _
   ValueName As String, vNewValue As String)

End Property
```

In altre parole vNewValue in Property Let, l'input della stringa, viene confrontato con l'output di stringa di Property Get.

6. Aggiungete le dichiarazioni per le API del registro nella sezione General del modulo di classe mostrato nel Listato 10.12. Notate che le dichiarazioni Public di costanti, variabili e funzioni esterne non sono consentite all'interno del modulo di classe. In alternativa si può aggiungere semplicemente i moduli delle dichiarazioni al progetto.

Listato 10.12 *Dichiarazione e costanti di API del registro.*

```
Option Explicit
'Costanti del Registro
Const HKEY_CLASSES_ROOT = &H80000000
Const HKEY_CURRENT_USER = &H80000001
Const HKEY_LOCAL_MACHINE = &H80000002
Const HKEY_USERS = &H80000003
Const REG_SZ = (1)          'Stringa Unicode con null alla fine
Const KEY_ALL_ACCESS = &H3F
Const ERROR_SUCCESS = 0&

Private Declare Function RegCloseKey Lib "advapi32.dll" _
  (ByVal hKey As Long) As Long

Private Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias _
     "RegOpenKeyExA" (ByVal hKey As Long, _
     ByVal lpSubKey As String, ByVal ulOptions As Long, _
     ByVal samDesired As Long, phkResult As Long) As Long

Private Declare Function RegQueryValueEx Lib "advapi32" _
    Alias "RegQueryValueExA" (ByVal hKey As Long, _
    ByVal lpValueName As String, ByVal lpReserved As Long, _
    ByRef lpType As Long, ByVal szData As String, _
    ByRef lpcbData As Long) As Long

Private Declare Function RegSetValueEx Lib "advapi32" _
    Alias "RegSetValueExA" (ByVal hKey As Long, _
    ByVal lpValueName As String, ByVal Reserved As Long, _
    ByVal dwType As Long, ByVal szData As String, _
    ByVal cbData As Long) As Long
```

7. Aggiungete il codice delle procedure Property Get Value e Property Let Value. Property Get deve essere passata con i parametri hKey, percorso di registro e nome del valore da recuperare. Viene restituito il valore come stringa. Property Let accetta gli stessi parametri di Property Get, con l'aggiunta del nuovo valore da impostare (vNewValue).

   La logica di queste procedure è ricavata dal progetto Values.Vbp ed è già stata illustrata in questo capitolo quando si è parlato del progetto in questione.

```
Public Property Get Value(hKey As Long, SubKey As String, _
    ValueName As String) As String
```

```
Dim szBuffer As String, dataBuff As String, _
   ldataBuffSize As Long, phkResult As Long, RetVal As Long
 'Crea Buffer
dataBuff = Space(255)
IdataBuffSize = Len(dataBuff)

   RetVal = RegOpenKeyEx(hKey, SubKey, 0, KEY_ALL_ACCESS, _
phkResult)
RetVal = RegQueryValueEx(phkResult, ValueName, 0, 0, dataBuff, _
   ldataBuffSize)
If RetVal = ERROR_SUCCESS Then
   Value = Left(dataBuff, ldataBuffSize - 1)
   'Elimina il terminatone!
Else
   MsgBox "Errore interno in RegQueryValueEx!"
End If
 'Chiude le chiavi
RegCloseKey hKey
RegCloseKey phkResult
End Property
```

Questo vale per Property Get Value. Vediamo la procedura Property Let Value:

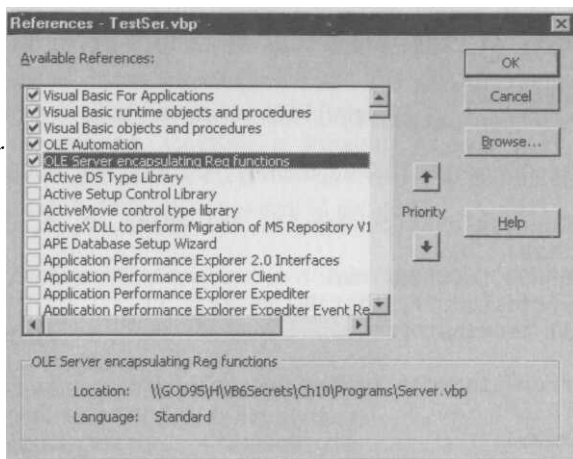
```
Public Property Let Value(hKey As Long, SubKey As String, _
   ValueName As String, vNewValue As String)
   Dim phkResult As Long, RetVal As Long
   RetVal = RegOpenKeyEx(hKey, SubKey, 0, KEY_ALL_ACCESS, _
   phkResult)
   RetVal = RegSetValueEx(phkResult, ValueName, 0, _
      REG_SZ, vNewValue, CLng(Len(vNewValue) +1))
   If RetVal <> ERROR_SUCCESS Then
      MsgBox "Impossibile scrivere nel Registro!"
   'Chiude le chiavi
   RegCloseKey hKey
   RegCloseKey phkResult
End Property
```

8. Eseguite l'applicazione server. L'applicazione che collauda questo server sarà posta in esecuzione in un'altra istanza di Visual Basic. In alternativa si può compilare il server, aprire un nuovo progetto e chiamare il server compilato "come se fosse vero".

Chiamare il server

Vediamo come si chiama il server:

1. Mediante l'istanza di VB che contiene il progetto del server, in esecuzione, aprite una nuova istanza di Visual Basic. Avviate un nuovo progetto. Nel dialogo *References* (selezionate *References* dal menu *Project),* attivate *OLE Server Encapsulating Reg Functions* (si veda la Figura 10.18); questa è la *Application Description* inserita nel passo 1 (si veda la Figura 10.15).

Figura 10.18 *Si utilizza il dialogo References per includere le classi del server OLE nel progetto corrente.*

2. In una routine progettata per recuperare i valori nel registro, dimensionate una nuova copia del modulo di classe del server ActiveX. Il riferimento nell'istruzione dell'istanza è nella forma: nome_applicazione.modulo_-classe.

   Richiamate la procedura `Property Get` con i parametri appropriati; per esempio:

```
Private Sub cmdGet_Click()
    Dim X As New ReadandWriteRegVals.RegData
    txtReturn.Text = X.Value(GethKey,
        txtSubKey.Text, txtValueName.Text)
End Sub
```

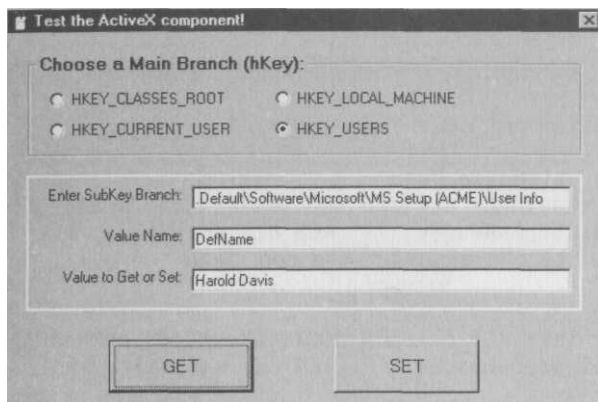
3. In modo analogo, create una routine per memorizzare i valori nel registro con una variabile che rappresenta una nuova istanza del modulo di classe del server ActiveX:

```
Private Sub cmdSet_Click()
    Dim X As New ReadandWriteRegVals.RegData
    X.Value(GethKey, txtSubKey.Text,
        txtValueName.Text) = txtReturn.Text
End Sub
```

Questo è veramente tutto quello che c'è da fare. Se creare e fare il debug di server OLE ActiveX può essere ovviamente abbastanza complicato (ne parleremo ulteriormente nel Capitolo 23), per chiamare metodi OLE esposti sono sufficienti alcune righe di codifica.

Si può obiettare che l'utilità di questa particolare applicazione ActiveX (server OLE) presenta alcune limitazioni. Da un lato, incapsula solo due API del registro (come mostrato in Figura 10.19); per di più, se si vuole utilizzare `Property Get Value` e `Property Let Value` occorre essere in grado di passare il corretto `hKey`, il percorso preciso del registro relativo al valore che si vuole ricercare o modificare, oltre al

nome corretto del valore stesso. Troppo esigente! Sembra ancora una tecnologia che mette soggezione.

Figura 10.19 *Ecco come si ricerca un valore nel registro mediante una chiamata dell'applicazione ActiveX.*

Registrazione delle estensioni dei file

Come argomento finale del capitolo, voglio illustrare come si registra l'estensione di un file. La registrazione accurata dell'estensione di un file consente l'esecuzione dell'applicazione appropriata quando si apre un file con quella estensione, di solito con il file aperto come argomento della riga di comando. Per esempio, a patto di avere Word installato sul proprio sistema, un doppio clic su un file .Doc in Gestione risorse fa partire Word e caricare il file in questione.

Nel programma esempio che stiamo per considerare, l'estensione di file .Bad viene associata a WeBe.Exe. Il codice sorgente del progetto è presente nel CD-ROM con il nome di FileX.Vbp, l'eseguibile è WeBe.Exe, il file di prova è Test.Bad (si tratta semplicemente di un file di testo . Txt creato in Notepad).

Per la dimostrazione completa di come funziona il progetto (in altre parole, per avviare WeBe.Exe con un doppio clic su Test.Bad), è necessario copiare il file WeBe.Exe nella directory principale C:\. Questo è dovuto al fatto che la riga di comando che apre l'applicazione è stata inserita nel Registro di configurazione come "C:\WeBe.Exe %1"; così è stata indicata in modo rigido nel codice del programma. Naturalmente nulla vieta di modificare la riga del comando di apertura nel codice sorgente del progetto e poi rimandarlo in esecuzione.
In un nuovo progetto, aggiungere il modulo di dichiarazioni API del registro, RegAPI.Bas. Successivamente, in base all'impostazione di avvio, sistemare la codifica illustrata nel Listato 10.13 nel modulo che contiene *Sub Main* oppure in corrispondenza dell'evento (e modulo) di caricamento del forni predefinito:

Listato 10.13 *Registrazione dell estensione di un file.*

```
Option Explicit
Public Const MAX_PATH = 256&

Public Sub Main()
  Dim sKeyName As String, sKeyValue As String, Retval As Long, _
     phkResult As Long
 'Crea una voce radice per WeBe
sKeyName = "WeBe"
sKeyValue = "File Extension Demo"
Retval = RegCreateKey(HKEY_CLASSES_ROOT, sKeyName, phkResult)
Retval = RegSetValue(phkResult, "", REG_SZ, sKeyValue, 0&)
 'Crea una voce radice che associa .Bad con "WeBe"
sKeyName = ".Bad"
sKeyValue = "WeBe"
Retval = RegCreateKey(HKEY_CLASSES_ROOT, sKeyName, phkResult)
Retval = RegSetValue(phkResult, "", REG_SZ, sKeyValue, 0&)

       'Imposta la riga di comando per WeBe
sKeyName = "WeBe"
sKeyValue = "C:\WeBe.Exe %1"
 'Cambia sKeyValue perché corrisponda
 'alla posizione effettiva dell'eseguibile!
Retval = RegCreateKey(HKEY_CLASSES_ROOT, sKeyName, phkResult)
Retval = RegSetValue(phkResult, "shell\open\command", _
   REG_S2, sKeyValue, MAX_PATH)
MsgBox ".Bad" 'Eseguite qualcosa per far vedere che il test ha funzionato!
End Sub
```

Se si esegue questo programma e poi si apre Regedit, si può notare che è stata creata una voce relativa all'estensione di file .Bad in corrispondenza di HKEY_-CLASSES_ROOT:

```
.Bad = WeBe
```

Inoltre, viene creata per WeBe una struttura per il comando di apertura nella shell (si veda la Figura 10.20).

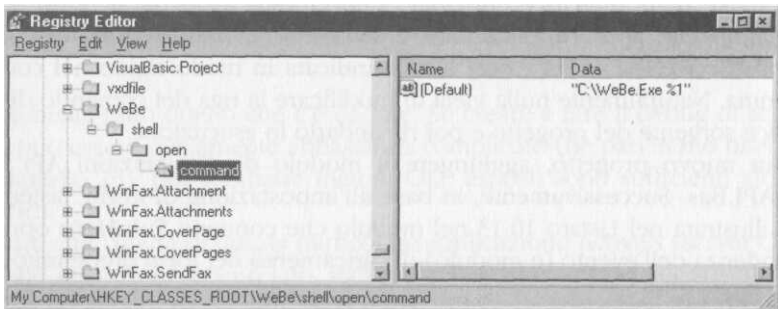
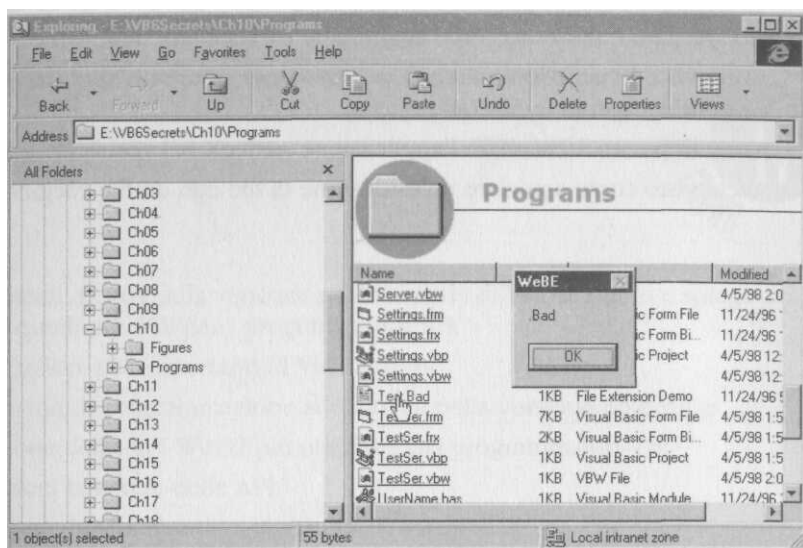
Figura 10.20 *E stata creata una struttura Shell OpenCommand che richiama Be.Exe per i file con estensione .Bad.*

Per verificare il funzionamento di quanto predisposto, compilate il programma per creare un file eseguibile di nome WeBe.Exe, Copiate il file nella directory C:\. In Gestione risorse, fare doppio clic su un file qualsiasi con estensione .Bad. Viene eseguito WeBe.Exe (si veda la Figura 10.21).

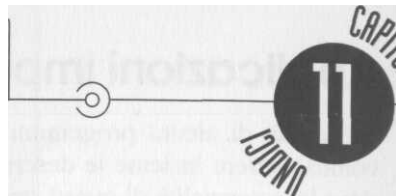
Figura 10.21
Un doppio clic su un nome di file con estensione Bad in Gestione risorse richiama l'applicazione WeBe.

In breve, il registro ora sa quale applicazione associare ai file .Bad.

Riepilogo

Tutto qua, quello che serve sapere per programmare il registro. Questo capitolo dovrebbe essere sufficiente per mettervi in grado di creare applicazioni che facciano un uso soddisfacente del registro.

- Vi ho mostrato le API del registro disponibili.
- Vi è stata data una descrizione di quello che fa ogni API.
- Vi ho mostrato come dichiarare le API del registro, le costanti e le strutture nei vostri progetti.
- Vi è stato dato un modulo .Bas con le dichiarazioni da utilizzare facilmente per il loro inserimento nei vostri progetti.
- Ho spiegato come passare da stringhe VB a stringhe in stile C con terminatore nullo, e viceversa.
- Avete scoperto come utilizzare le quattro istruzioni del registro incorForate in VB.
- Avete imparato a ricercare e visualizzare chiavi e sottochiavi.

- Avete imparato a modificare i valori e i diversi modi per rilevare e memorizzare le informazioni sull'utente.
- Ho descritto l'API GetUserName, compresa la differenza tra dove questa funzione legge il suo valore nel registro e dove le routine di inizializzazione di Microsoft leggono le loro.
- Ho spiegato e illustrato come aggiungere e cancellare delle voci.
- Avete visto le istruzioni passo dopo passo per creare un'applicazione ActiveX che incapsula le API del registro.
- Avete imparato a chiamare l'applicazione ActiveX nel vostro progetto.
- Avete visto come associare un'estensione di file con un file eseguibile.

VISUAL STUDIO API WIN32 E MESSAGGI

- Gli strumenti inclusi nella versione professionale di Visual Studio e spiegazione di quelli più utili per i programmatori VB.
- Le applicazioni più imFortanti di Visual Studio.
- Gli strumenti di programmazione addizionali nella versione Enterprise.
- Illustrazione delle API Win32 più utilizzate dai programmatori VB.
- Applicazioni concrete delle API.
- Spiegazione del sistema di messaggi di Windows.
- Aggiunta di menu di scelta rapida alle caselle di testo.
- Intercettazione dei messaggi di Windows.
- Sistemazione di icone delle applicazioni nel "vassoio" di Windows.

Questo capitolo si occupa degli strumenti di Visual Studio versione Professional e Enterprise, di API Win32, del sistema di messaggi di Windows e di alcuni argomenti collegati a questi. Nel caso in cui le vostre finalità di sviluppo andassero oltre quello che si può fare con Visual Basic, che di per se stesso ha dimensioni ragguardevoli, dovrete avere a che fare con gli strumenti e le tecniche spiegate in questo capitolo. In generale, questa eventualità ha luogo quando il progetto di sviluppo deve interagire con parti del sistema operativo di Windows.

Gli strumenti che prima erano inclusi nel Win32 Software Development Kit (SDK) sono ora presenti in Visual Studio 6; vi si può accedere dalla voce di menu dei programmi comuni in Visual Studio.

Strumenti di Visual Studio 6.0 versione Professional

Visual Studio 6.0 Professional Edition esce accompagnato con una ricca collezione di strumenti la cui utilità e stile d'impiego vanno dal sublime al banale. La collocazione fondamentale di questi strumenti rimane l'ambiente di sviluppo Visual C++, anche se molti di questi possono essere imFortanti per chi sviluppa in Visual Basic.

Sfortunatamente, tra la documentazione inclusa con il prodotto non esiste un elenco completo di quello che i programmi forniti insieme a Visual Studio Tools possono fare. La documentazione relativa ai Tools OLE che fanno parte della collezione di utility di Visual Studio può essere studiata selezionando l'icona OLE Tools dal menu di programma di Visual Studio Tools.

Applicazioni importanti di Visual Studio

La finalità di alcuni programmi può non risultare chiara; per questo motivo ho voluto mettere insieme le descrizioni che seguono. Allo scopo di sfruttare al massimo le potenzialità di questi strumenti, questo capitolo è dedicato alla descrizione di quelli più imFortanti. Per avere maggiori informazioni su queste applicazioni, si provi a utilizzare il sistema di guida dell'applicazione (se disponibile), oppure si apra l'applicazione e si provi ad utilizzarla.

- **ActiveX Control Test Container,** come suggerito dal nome, riguarda la verifica dei controlli ActiveX. Questa applicazione consente di vedere che cosa succede quando si interviene su un controllo, se ne modificano le proprietà e si attivano i suoi eventi. Se si utilizza VBScript, è possibile rendere automatico un protocollo di verifica del controllo all'interno dell'applicazione ActiveX Control Test Container. Per avere maggiori informazioni si veda la Parte Sesta del libro.

  Una volta avviata l'applicazione, è possibile lanciare un controllo da verificare selezionando *Insert New Control* dal menu *Edit;* per esempio, Calendar Control 8.0. Quando si agisce sul calendario cambiando mese, anno o la data, è possibile osservare l'innesco di una serie di notifiche che riguardano gli eventi e le modifiche delle proprietà, nel pannello inferiore di Test Container come mostrato in Figura 11.1.

  Per evidenziare i fogli proprietà del controllo, come mostrato in Figura 11.2, una volta selezionato il controllo occorre scegliere *Properties* dal menu *Edit.* Quando vengono effettuate modifiche nel dialogo *Properties,* queste si ripercuotono nel modo in cui il controllo compare nel pannello superiore di ActiveX Control Test Container, mentre le proprietà modificate sono mostrate nel pannello inferiore (si veda la Figura 11.3).

Figura 11.1
Si può utilizzare ActiveX Control Test Container per esaminare le modifiche di eventi e proprietà nella elaborazione di un controllo.

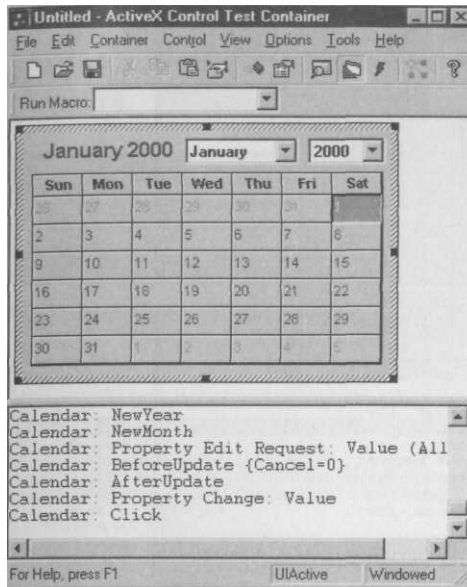



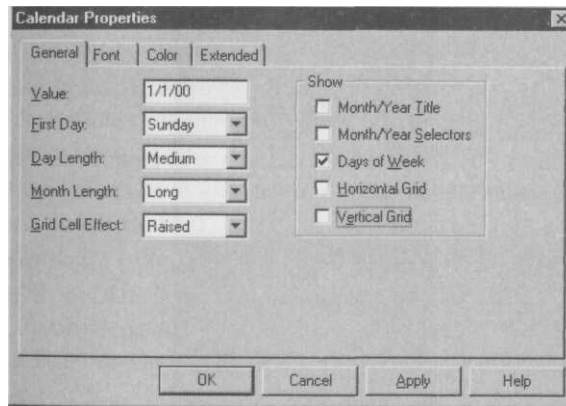
Figura 11.2
Si possono modificare le proprietà di un controllo in prova e vedere quello che succede.

Una volta selezionato il controllo, per eseguire o richiamare un metodo del controllo, selezionare *Invoke Methods* dal menu *Control.* Si può utilizzare il dialogo *invoke Metbods,* mostrato in Figura 11.4, per selezionare un metodo e i suoi parametri.

Figura 11.3
Le modifiche delle proprietà vengono visualizzate in ActiveX Control Test Container.

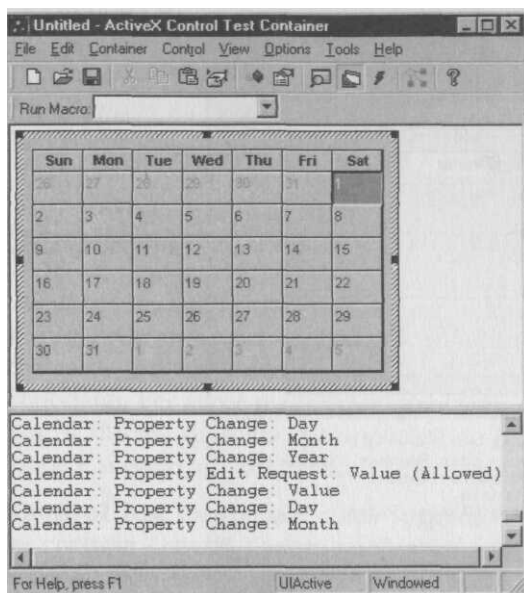
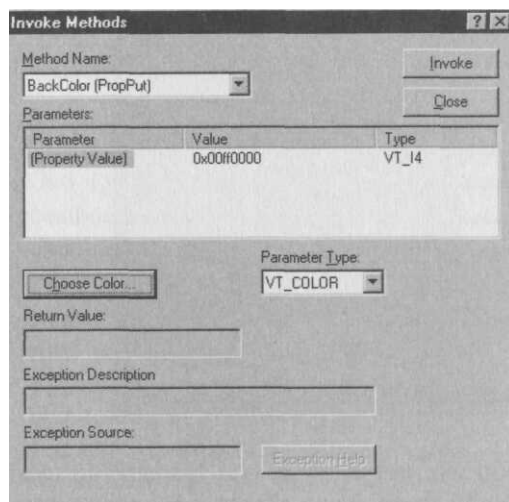
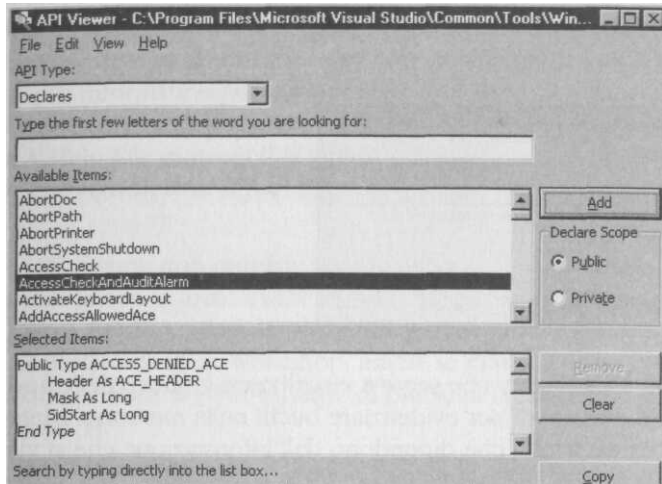



Figura 11.4
Si utilizza il dialogo Invoke Methods per eseguire i metodi del controllo in prova.

Selezionate Macros *dal menu* Tools *per aprire una macro VBScript che effettua in modo automatico la verifica del controllo.*

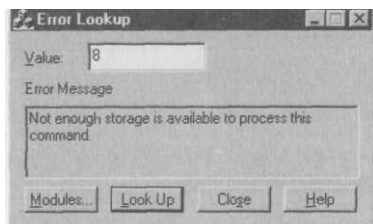
***API* Text Viewer,** mostrata nella Figura 11.5, viene *utilizzata,* per copiare e incollare nelle applicazioni costanti, dichiarazioni e tipi relativi ad API di Windows. Per avere maggiori informazioni sull'utilizzo di questa utility, si veda il Capitolo 4.

Figura 11.5 *Si può utilizzare API Text Viewer per aggiungere* **in** *modo preciso dichiarazioni API nelle applicazioni.*

- **AVI Editor** (Editor di file AVI) è un'utility per la riproduzione e la modifica di file multimediali .Avi. Può essere utilizzata per preparare e verificare catture multimediali dello schermo da inserire in un tutorial online.
- **DataObject Viewer** (Visualizzatore Data Object) viene utilizzata per osservare gli oggetti della Clipboard e gli oggetti che possono essere trasFortati mediante drag and drop.
- DDE Spy viene utilizzata per tenere traccia e visualizzare messaggi e caliback DDE. Il DDE è un vecchio protocollo che serve a scambiare dati fra oggetti e non viene quasi più utilizzato, anche se non si può mai dire. Se si ha a che fare con messaggi DDE, questa utility calza a pennello.
- **Depends,** abbreviazione di Dependency Walker (Dipende, in italiano), è una utility che mette a disposizione un elenco di tutti i moduli, per esempio i file delle librerie DLL, richiesti da una particolare applicazione. Inoltre, fornisce molte altre informazioni utili per la risoluzione di problemi, tra le quali un elenco di tutte le funzioni esFortate da un modulo, dei file mancanti, dei file danneggiati e dei file che sono stati compilati per un tipo di macchina sbagliato.

DocFile Viewer (Visualizzatore DocFile) visualizza il contenuto e la struttura di un documento OLE composto. Una volta che il file del documento è stato caricato, DocFile ne visualizza il contenuto utilizzando due tipi di finestre: una riguarda la struttura gerarchica della memorizzazione e l'altra visualizza il contenuto del flusso (stream).

Se siete interessati agli strumenti che consentono di lavorare con i documenti OLE composti, potete rivolgervi alle spiegazioni del Capitolo 28 relative alle applicazioni ActiveX Documents, che sono la versione Visual Basic dei documenti OLE composti.

- **Error Lookup** (Ricerca errori), mostrato in Figura 11.6, traduce codici di errore del sistema operativo o del programma nei corrispondenti messaggi di testo.

Figura 11.6 *L'utility Error Lookup serve a individuare il significato dei codici di errore.*

HeapWalk è una utility che serve a visualizzare gli heap delle applicazioni Win32 e ad esaminarli per evidenziare buchi nella memoria. L'interfaccia è costituita da tre livelli, che dipendono dall'informazione che si vuole visualizzare:

• La finestra di visualizzazione *Summary* mostra un elenco degli heap relativi a tutte le applicazioni Win32 che sono in esecuzione nel sistema. La Figura 11.7 mostra la finestra in vista *Summary* di HeapWalk.

Process Id	Module Name	Heap Id	Bytes Alloc	Obj Count	Bytes Free
fff0bc6d	VB5.EXE	7d776f19	0	0	1048428
fff0bc6d	VB5.EXE	7cc86f19	120	4	1048292
fff0bc6d	VB5.EXE	7cd86f19	720	30	1047588
fff0bc6d	VB5.EXE	7c2d6f19	264	9	1048128
fff0bc6d	VB5.EXE	7a9c6f19	1976	9	1046416
fff0e01d	MMTASK.TSK	7ee86f19	3896	20	1044364
fff0f1b9	EXPLORER.EXE	7eec6f19	88628	1759	955860
fff15405	IOWATCH.EXE	7eea6f19	0	0	1052524
fff1586d	IMGICON.EXE	7eea6f19	0	0	1052524
fff17009	CAPTURE.EXE	7ee96f19	0	0	1048428
fff24429	WINWORD.EXE	7ee86f19	572248	864	476272
ffffa22d	MPREXE.EXE	7ef96f19	0	0	1052448
ffffc5a1	MSGSRV32.EXE	7ee86f19	65880	5	982504
ffffc5a1	MSGSRV32.EXE	7ef86f19	3692	16	1048756
ffffdb35	SPOOL32.EXE	7ee96f19	36	3	1052400

Figura 11.7 *Si può utilizzare HeapWalk per osservare gli heap assegnati alle applicazioni a 32 bit e verificare gli eventuali buchi.*

• La finestra di visualizzazione *Details* mostra tutti gli oggetti di memoria dello heap selezionato.

• La finestra di visualizzazione *Object* mostra il blocco di memoria allocato per l'oggetto considerato. Vengono cercati i buchi di memoria riprendendo delle istantanee dalla finestra *Details,* prima e dopo l'esecuzione di una certa operazione. Gli oggetti di memoria che esistono nell'istantanea scattata dopo la conclusione di un'operazione sono potenzialmente dei buchi di memoria che possono quindi essere esaminati più a fondo.

Help Workshop mette a disposizione un insieme completo di strumenti in un ambiente di lavoro visuale per la creazione e la modifica di sistemi di guida. Si può notare che Help Workshop fornisce un comodo accesso a tutte le API WinHelp.

Oltre a tutto questo, è possibile utilizzare Help Workshop per stabilire facilmente l'ID di un argomento e il file sorgente. Se è attivo il comando Help Author *nel menu* File e si *fa clic destro su un argomento qualsiasi (compresi i menu a discesa), è possibile fare clic su* Topic Information *per visualizzare una serie di informazioni legate a quell'argomento. Se era attiva la casella* Include .Rtf Filename And Topic ID *nel dialogo* Compile A Help File *quando il file di aiuto è stato compilato, è possibile vedere informazioni che riguardano il file .Rtf, che include l'argomento e l'ID dell'argomento.*

Anche se Help Workshop non prevede tutte le funzioni di alcuni programmi di terze parti per la preparazione di aiuti, a differenza di questi è completamente gratuito (se si possiede Visual Studio). Vista la tendenza attuale di sviluppare guide in stile HTML, non supFortate da Help Workshop, anche se gratuito il suo utilizzo può non essere più molto interessante. Per saperne di più sulla creazione di sistemi di guida si veda il Capitolo 34.

- **OLE View,** o OLE/COM Object Viewer, facilita la creazione di applicazioni OLE e COM migliorando la comprensione di quello che succede nei rispettivi programmi. È anche un potente strumento di collaudo che consente di controllare il comFortamento previsto da oggetti e interfacce. Dato che la fonte principale delle informazioni di questo strumento è il Registro, si può utilizzare Object Viewer anche per verificare l'accuratezza delle informazioni di registrazione OLE. OLE Object Viewer è per gli sviluppatori che vogliono trovare risposte a domande come le seguenti:
- Quali oggetti OLE/COM risultano installati sul mio sistema? (Questa informazione è disponibile nel pannello di sinistra).
- Quali interfacce sono supFortate da un determinato oggetto? (Questa informazione è disponibile nel pannello di destra).
- L'oggetto considerato è un server o un gestore in esecuzione, oppure è locale? Quali voci lo riguardano nel database di registrazione? (Questa informazione è disponibile nelle schede *Registration* e *Implementation* nel pannello di destra).
- **Package and Development Wizard** (Creazione guidata pacchetti di installazione), il wizard già noto come Setup, è una utility potente e multifunzionale che può gestire installazioni tradizionali e via Internet (si veda la Figura 11.8). Per avere maggiori informazioni sui programmi di installazione, si veda il Capitolo 35.
- **Process Viewer** (Visualizzatore processo) può essere utilizzato per identificare processi e thread in esecuzione, come mostrato in Figura 11.9.

Figura 11.8

// wizard Package and Development consente di creare e gestire diversi tipi di routine di installazione.

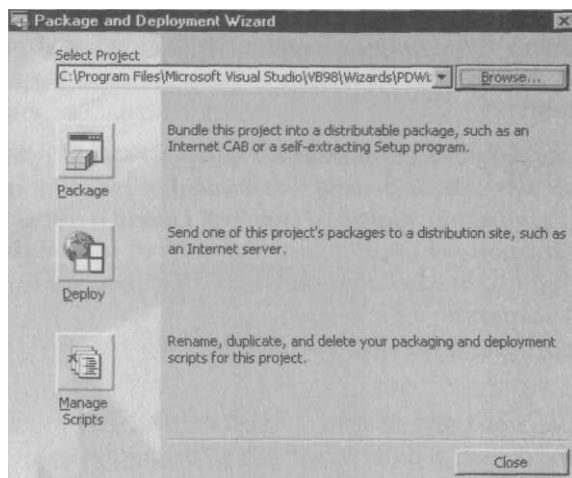
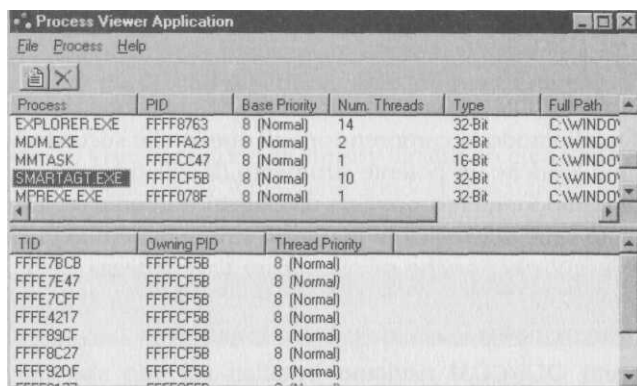



Figura 11.9

Si può utilizzare Process Viewer per analizzare processi e thread in esecuzione.

- **ROT Viewer** (Visualizzatore Rot) visualizza in modo dinamico il contenuto della tabella relativa a un oggetto OLE in esecuzione.
- Spy++ è uno strumento Form1dabile per indagare il contenuto del flusso di messaggi di Windows; si possono trovare informazioni sul sistema di messaggi di Windows più avanti in questo capitolo. Grazie alla versione 6 i programmatori VB possono utilizzare il subclassing (e le relative tecniche) per intercettare i messaggi di Windows che vengono inviati a un modulo o a un controllo. Conoscere i messaggi inviati sulla base di particolari azioni dell'utente è una questione imFortante per i programmatori VB, che si può svolgere facilmente utilizzando Spy.

  Spy consente di selezionare la finestra per la quale si vogliono visualizzare i messaggi; si possono anche selezionare tutte le finestre, se lo si desidera. È possibile inviare l'output su una finestra dello schermo che, a sua volta, può avere effetto sui messaggi visualizzati, oppure a un file o su una Forta seriale. Per esempio, se si fa partire un progetto VB con un Form1 di avvio

predefinito, mediante la finestra *Select* di Spy è possibile selezionare Form1 come selezione corrente (si veda la Figura 11.10). La finestra *Messages* mostra quindi i messaggi ricevuti dal form. A questo proposito, si può notare che il nome di classe per un form VB rimane ThunderFormDC; divertente, vero?

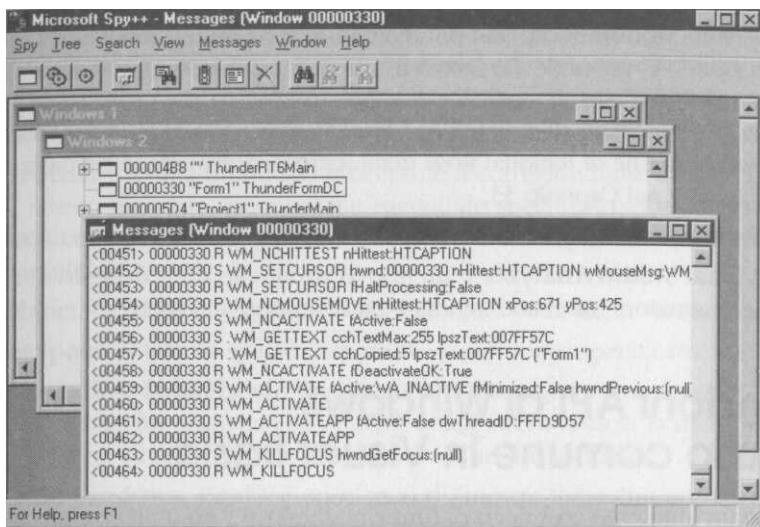
Figura 1MU
L'utility Spy consente di osservare il messaggio di Windows collegato al modulo Visual Basic.

Stress Utility consente di saturare determinate risorse del sistema, in modo da collaudare le applicazioni in condizione estreme, di risorse ridotte al minimo. Si possono saturare le seguenti risorse: lo heap globale, lo heap utente, lo heap GDI, lo spazio su disco e i gestori dei file. L'icona di questa utility non poteva che essere quella di un piccolo elefante (l'icona è visibile in Figura 11.11).

Figura 11.11
Sipuò utilizzare Zoomin.Exe per catturare una parte detto schermo.

WinDiff viene utilizzato per confrontare graficamente il contenuto di due file o di due directory.

Zoomin (Zoom avanti) consente di catturare una parte dello schermo (si veda la Figura 11.11). Questa operazione è utile perché si può copiare il contenuto di Zoomln negli Appunti e incollarlo poi in una utility di gestione delle immagini, dove può essere utilizzato come parte di una icona o come bitmap.

Strumenti di Visual Studio 6.0 Enterprise

Oltre agli strumenti previsti dalla versione Professional, Visual Studio 6.0 Enterprise viene distribuito con i seguenti strumenti:

- **Application Performance Explorer** serve ad analizzare le prestazioni client/server.
- **Visual Modeler** consente di creare in modo visivo il modello di un oggetto astratto. È possibile far generare automaticamente il codice a partire da questo modello. Il codice viene poi inserito in un progetto VB che può essere "riFortato all'indietro" in Visual Modeler. Visual Modeler costituisce un sotto insieme di Rational Rose della Rational Software; se ne parla più diffusamente nel Capitolo 33.
- **RemAuto Connection Manager** configura l'accesso a server remoti.
- **Visual Studio Analyzer** è un nuovo strumento che consente di analizzare le interazioni tra eventi generati dai componenti remoti.

Funzioni API di Windows di uso comune in Visual Basic

VB6è un ambiente così ricco di funzioni che lo si può utilizzare per scrivere applicazioni complesse che non devono mai chiamare API di Windows. Tuttavia, se si vuole dare libero sfogo alla piena potenza di Windows e quindi estendere VB quasi all'infinito, le API sono a volte necessarie, o quanto meno consigliabili. Di solito le chiamate che le riguardano rientrano in una delle quattro categorie seguenti:

- Funzioni di registro, quando si richiede una ricerca e una manipolazione del registro che va oltre le possibilità consentite dalle istruzioni incorForate in VB (si veda il Capitolo 10).
- Funzioni di informazioni sul sistema.
- Funzioni grafiche e di visualizzazione.
- Funzioni per la ricerca di informazioni non disponibili in VB, come le funzioni *Message.*

Vediamo alcune delle API utilizzate più comunemente dai programmatori VB, a esclusione delle API del registro che sono state trattate diffusamente nel Capitolo 10, insieme a una breve descrizione di quello che fanno.

Devo dire per esperienza che non è sempre facile comprendere il funzionamento delle API con un solo esempio; questo è il motivo per cui ho incluso nella prossima sezione di questo capitolo, diversi esempi i quali utilizzano molte delle API qui descritte.

- BitBlt. Sposta una bitmap da un contesto di dispositivo sorgente verso una destinazione; viene spesso utilizzato per elaborare il fattore di scala, la dimensione e l'aspetto di immagini bitmap.

- **BringWindowToTop.** Forta una finestra in primo piano (in cima all'ordine stabilito da ZOrder) e la rende attiva; si veda SetActiveWindow.
- **ClipCursor.** Confina il cursore in una zona rettangolare dello schermo.
- **CreateCompatibleDC.** Prepara un'immagine nella memoria, come per esempio prima della copia di un'immagine su un dispositivo compatibile.
- **CreateCursor, DestroyCursor** CreateCursor genera un cursore caratterizzato da dimensione specificata, modello di bit e zona attiva. DestroyCursor elimina un cursore creato dalla funzione CreateCursor e libera la memoria che questo occupava. (Non si deve utilizzare questa funzione per eliminare un cursore che non è stato creato con la funzione CreateCursor).
- **CreateProcess.** Genera un nuovo processo e il corrispondente thread primario. Il nuovo processo esegue il file eseguibile specificato. (Questa funzione *rimpiazza,* WinExec la quale tuttavia esiste ancora per una questione di compatibilita.)
- DeleteObject Cancella un oggetto dalla memoria; si veda anche ReleaseDC.
- **DiskFreeSpace.** Rileva lo spazio disponibile su un disco specificato.
- **DragAcceptFiles, DragFinish.** DragAcceptFiles registra una finestra in grado di accettare file trascinati su di essa; DragFinish rilascia la memoria che Windows ha allocato per il trasferimento di un file trascinato.
- **EnumChildWindows.** Produce un elenco di finestre figlie che appartengono alla finestra genitore specificata.
- **ExtFloodFill.** Riempie una zona della superficie di visualizzazione con il pennello corrente.
- **ExtractIcon, DrawIcon, LoadIcon.** Queste funzioni elaborano le icone.
- **FillRect.** Riempie un rettangolo con il pennello corrente.
- **FindExecutable.** Ricerca e recupera nome e handle dell'eseguibile associato al nome del file specificato.
- **GetActiveWindow.** Recupera l'handle della finestra attiva associato con il thread che richiama la funzione.
- **GetCursorPos.** Rileva la posizione del cursore, espressa secondo le coordinate dello schermo.
- **GetDesktopWindow.** Recupera un handle alla finestra della scrivania di Windows, che ricopre l'intero schermo ed è l'area sopra la quale vengono disegnate tutte le icone e le altre finestre.
- **GetDiskFreeSpace.** Restituisce informazioni su un disco, inclusa la quantità di spazio libero.
- **GetModuleFileName.** Restituisce il percorso completo e il nome del file relativi all'eseguibile che contiene un modulo specificato. (Quando si esegue con Windows a 32 bit, la funzione riForta nomi lunghi dei file, se disponibili e se il numero di versione dell'applicazione è superiore o uguale a 4.00. Se queste condizioni non sono verificate, riForta i nomi dei file nel consueto formato 8.3.)

- **GetPaletteEntries, CreatePen, SelectObject.** Queste funzioni elaborano palette di colori.
- **GetParent.** Prende l'handle del genitore di una finestra.
- **GetSystemDirectory.** Prende il percorso della directory di sistema di Windows. Le applicazioni non devono creare file nella directory di sistema. Se l'utente sta lavorando con una versione condivisa di Windows, l'applicazione non dispone di accesso in scrittura alla directory di sistema. Le applicazioni devono creare file solo nella directory indicata dalla funzione GetWindowsDirectory.
- **GetSystem1nfo.** RiForta informazioni sul sistema corrente. Questa funzione sostituisce GetWinFlags, che risulta obsoleta e non viene più utilizzata dalle API Win 32.
- **GetSystemMenu.** Consente l'accesso al menu *Window* per la copia e la modifica di voci del menu di sistema. (Si apre questo menu facendo clic sull'icona del modulo nella parte sinistra superiore di un modulo; a volte è chiamato menu di sistema o menu di controllo.)
- **GetSystemMetrics.** Rileva la metrica del sistema, che riguarda le dimensioni (larghezza e altezza) degli elementi visualizzati da Windows. Tutte le dimensioni riFortate da GetSystemMetrics sono espresse in pixel.
- **GetVersionEx.** Restituisce informazioni sulla versione del sistema operativo in esecuzione (sostituisce GetVersion).
- **GetWindowLong, SetWindowLong.** Rileva oppure imposta informazioni sullo stile di una finestra.
- **GetWindowPlacement, SetWindowPlacement** Rileva oppure imposta lo stato di visualizzazione e le posizioni normale (di ripristino), minima e massima di una finestra.
- **GetWindowRect.** Rileva le dimensioni della finestra specificata, espresse in coordinate di schermo.
- **GetWindowsDirectory.** Rileva il percorso della directory di Windows; si veda GetSystemDirectory.
- **GetWindowText.** Rileva il titolo di una finestra oppure il testo di un controllo. GetWindowText non è in grado di rilevare il testo di un controllo situato in un'altra applicazione.
- **GetWindowTextLength.** Recupera la lunghezza in caratteri del testo relativo alla barra del titolo di una finestra (se la finestra prevede una barra del titolo). Se la finestra è un controllo, la funzione riForta la lunghezza del testo all'interno del controllo.
- **GlobalMemoryStatus.** Recupera informazioni sulla memoria attualmente disponibile. La funzione restituisce informazioni sulla memoria fisica e su quella virtuale.
- **IsIconic, IsWindowVisible, IsZoomed.** Determina lo stato di una finestra, ovvero stabilisce se è minimizzata, visibile oppure massimizzata.

- **LoadCursor.** Carica la risorsa cursore specificata dall'eseguibile dell'applicazione.
- **PlaySound.** Esegue un suono specificato mediante il nome di file, la risorsa oppure un evento di sistema.
- **ReleaseDC.** Cancella dalla memoria un contesto di dispositivo; si veda anche DeleteObject.
- **RemoveMenu.** Cancella una voce di menu.
- **RoundRect.** Disegna un rettangolo con gli angoli arrotondati. Il rettangolo viene tracciato con la penna corrente ed è riempito con il pennello corrente.
- **SendMessage.** Invia un messaggio in una finestra. Per esempio, il messaggio `WM_PAINT` dice a una procedura di finestra che l'area client della finestra è stata modificata e deve essere ridisegnata. Un altro esempio: nessuna proprietà incorForata in VB è in grado di dire quante righe sono contenute in una casella di testo multiriga; tuttavia, se si utilizza `SendMessage` per inviare un messaggio `EM_GETLINECOUNT` alla casella di testo, viene riFortato il numero di righe del testo contenuto.

  Si possono utilizzare molti altri messaggi, oppure se ne possono creare di propri. Per maggiori informazioni si veda più avanti in questo capitolo.
- **SetActiveWindow.** Rende attiva una finestra; si veda `BringWindowToTop`.
- **SetCapture.** Invia tutti gli input del mouse alla finestra specificata che appartiene al thread corrente, senza tenere conto della posizione del cursore all'interno della finestra. Solo una finestra per volta può catturare il mouse. Se il cursore del mouse si trova sopra una finestra creata da un altro thread, il sistema dirige l'input del mouse verso la finestra specificata solo se è stato premuto un pulsante del mouse.
- **SetCursorPos.** Sposta il cursore nella posizione specificata.
- **SetWindowPos.** Modifica dimensione, posizione e ZOrder (l'ordine tridimensionale con il quale vengono sistemati gli oggetti) relativi a una finestra figlia, popup o di massimo livello.
- **WinHelp.** Avvia Winhelp.Exe, l'applicazione di guida di Windows. È possibile aprire la guida su un file e un argomento specifici.

Utilizzo di API nel concreto

In questa sezione è mia intenzione mostrare alcuni impieghi concreti delle API in applicazioni Visual Basic. Si tratta di impieghi immediati, in quanto ho inserito ogni API in una propria procedura. Le applicazioni si trovano nel CD-ROM allegato al libro nella directory dei programmi relativi al Capitolo 11. È stata mia preoccupazione pensare a un facile inserimento di queste routine nei vostri progetti. Per facilitare questa operazione, nel CD-ROM sono presenti due moduli di codice:

- ***APIDec.Bas** include le dichiarazioni relative a molte funzioni API, a strutture e costanti utilizzate negli esempi di questo capitolo.*

- ***APICode.Bas*** *contiene il codice generalizzato che utilizza le diverse funzioni API, illustrate negli esempi di questo capitolo.*

A volte è possibile realizzare una stessa cosa in due modi diversi: mediante un controllo ActiveX oppure attraverso un 'API di Windows. Dato che la programmazione API tende a essere piuttosto complessa, perché uno dovrebbe scegliere questa strada? Una delle ragioni è legata al fatto che un programma che utilizza solo API, e nessun controllo, risulta probabilmente più piccolo e consuma meno risorse.

Sistemazione di un modulo in primo piano

Per sistemare un modulo in primo piano, in altre parole per farlo apparire sopra tutte le altre finestre, anche se non è attivo, come se il suo ZOrder fosse sempre a O, si richiama PAPI SetWindowsPos. A SetWindowsPos devono essere passati Phandle del form, la costante HWND_TOPMOST e i flag appropriati. Il Listato 11.1, che contiene le dichiarazioni per SetWindowsPos e le costanti relative, è ricavato dal file API-Dec.Bas:

Listato 11.1 *Dichiarazioni e costanti di SetWindowPos.*

```
Option Explicit
Declare Function SetWindowPos Lib "user32" _
   (ByVal hwnd As Long, ByVal hWndInsertAfter As Long, _
   ByVal x As Long, ByVal y As Long, ByVal ex As Long, _
   ByVal cy As Long, ByVal wFlags As Long) As Long
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOSIZE = &H1
Public Const SWP_SHOWWINDOW = &H40
Public Const SWP_NOACTIVATE = &H10
Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
```

Il codice seguente, Sub FormOnTop (ricavato dal file APICode.Bas), è una routine generica che definisce il form il cui handle è stato passato alla procedura in modo che sia sempre in primo piano (oppure no), in funzione del valore booleano passato insieme alla routine. Per esempio, è possibile chiamare FormOnTop dall'evento di disegno di un form per essere sicuri che questo sia sempre in primo piano:

```
FormOnTop Me.hWnd, TRUE
```

Si può anche chiamare FormOnTop con l'handle di un altro form come argomento:

```
FormOnTop frmOnTop.hWnd, FALSE
```

Il Listato 11.2 mostra la procedura FormOnTop.

Listato 11.2 *Definizione di una finestra in primo piano.*

```
Public Sub FormOnTop(Handle As Long, OnTop As Boolean)
 Dim  wFlags As Long,  PosFlag As Long
   wFlags = SWP_NOMOVE Or SWP_NOSIZE Or _
      SWP_SHOWWINDOW Or SWP_NOACTIVATE
   Select Case OnTop
      Case True
         PosFlag = HWND_TOPMOST
      Case False
         PosFlag = HWND_NOTOPMOST
   End Select
   SetWindowPos Handle, PosFlag, 0, 0, 0, 0, wFlags
End Sub
```

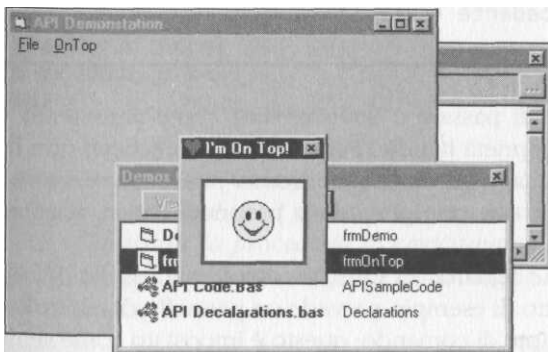
Nel programma dimostrativo sul CD-ROM, FormOnTop *viene chiamato da due differenti voci del menu. Una voce imposta il form corrente in modo che sia sempre in primo piano, utilizzando* Me.hWnd; *l'altra imposta* frmOnTop *che, ovviamente, non è il form su cui si trova il menu. La voce del menu che rappresenta un form ha il segno di spunta quando il modulo è sempre in primo piano, altrimenti non ce l'ha.*

Se si utilizza l'evento clic del menu per consentire all'utente di commutare uno stato, è facile verificare lo stato corrente utilizzando la proprietà .Checked *del menu.*

Vediamo il codice che definisce questo fatto per frmOnTop:

```
Private Sub mnuOnTopOtherForm_Click()
   If Not mnuOnTopOtherForm.Checked Then
   'attiva in primo piano
      FormOnTop frmOnTop.hwnd, True
      mnuOnTopOtherForm.Checked = True
      mnuOnTopThisForm.Checked = False
   Else  'disattiva
      FormOnTop frmOnTop.hwnd, False
      mnuOnTopOtherForm.Checked = False
   End If
End Sub
```

Davvero facile e divertente (si veda la Figura 11.12)!

Figura 11.12 *La faccina che sorride è sempre in primo piano ancbe quando non è attiva!*

Cosa succede se si impostano più finestre sempre in primo piano, utilizzando SetWindowsPos? Bella domanda. Pensavo che non lo avreste mai chiesto. Le finestre in primo piano stanno al di sopra di tutte le altre finestre, ma possono essere posizionate una sopra o sotto l'altra. In altre parole, le finestre sempre in primo piano si comFortano esattamente come le finestre normali, ma solo in riferimento l'una all'altra.

Spostamento dei controlli tra i form

La funzione SetParent può essere *utilizzata,* per spostare un controllo da un modulo a un altro; in questo modo si possono risparmiare tempo e risorse. In un programma che *utilizza una* grande casella di riepilogo in sei dialoghi diversi, è possibile predisForre una sola casella di riepilogo che si sposta nel dialogo opFortuno quando serve, anziché avere sei diversi controlli a casella di riepilogo. In questo modo non si deve riempire la casella di riepilogo ogni volta che serve.
Vediamo come si lavora. La dichiarazione di SetParent, *ricavata da* APIDec.Bas, è la seguente:

```
Declare Function SetParent Lib "user32"
    (ByVal hWndChild As Long, ByVal hWndNewParent As Long)
    As Long
```

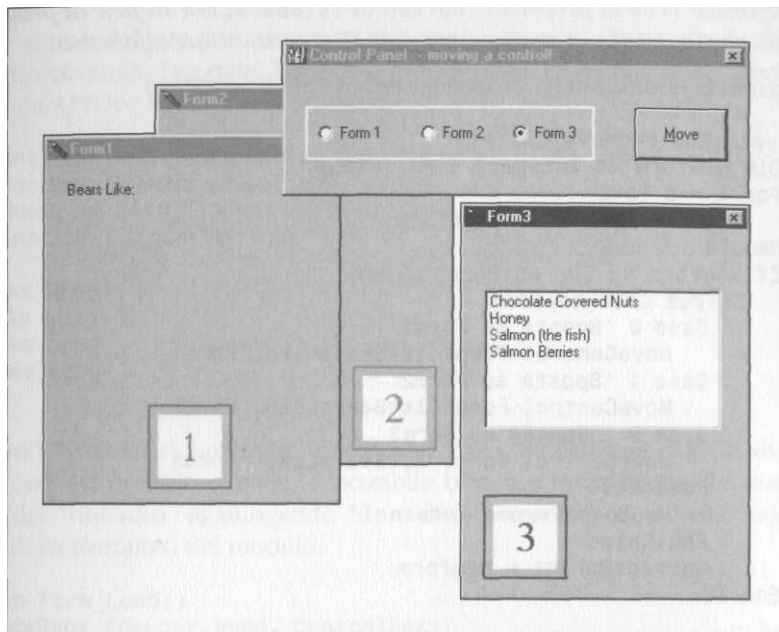
Si devono semplicemente passare a SetParent l'handle del controllo figlio e quello del form che si desidera avere come suo nuovo genitore (l'adozione via API!). A proposito, SetParent restituisce l'handle del vecchio genitore, se lo si desidera.
Per il modulo APICode.Bas ho generalizzato un po' tutto questo. Alla mia procedura MoveControl vengono passati un controllo (che deve essere spostato) e un form (che sarà il nuovo genitore). MoveControl poi chiama l'API SetParent con la proprietà .hWnd di ciascuno, come si vede nel Listato 11.3.

Listato 11.3 *Spostamento di un controllo in un nuovo genitore.*

```
Public Sub MoveControl(Child As Control, NParent As Form)
   Dim RetVal As Long
   RetVal = SetParent(Child.hWnd, NParent.hWnd)
   'Se necessario potrebbe passare l'handle
   'del genitore precedente (RetVal)!
End Sub
```

Qui si corre il pericolo di passare a MoveControl come argomento un controllo figlio che non ha una proprietà handle. Per esempio, le etichette non hanno la proprietà .hWnd. Se si passa a MoveControl come primo argomento un'etichetta, oppure un altro controllo che non prevede la proprietà .hWnd, si genera un errore in runtime.
Vediamo un esempio che riguarda lo spostamento di una casella di riepilogo tra tre diversi moduli. Il progetto di esempio prevede un pannello di controllo con tre pulsanti opzione e un pulsante di comando; questo è impostato come sempre in primo

piano grazie alla routine esaminata nella sezione precedente. Il progetto comprende anche tre form: Form1, Form2 e Form3. Form1 viene caricato con la casella di riepilogo lstBearsLike riempita con alcune voci (si veda la Figura 11.3). La demo sta lstBearsLike di modulo in modulo; è probabile che questo figlio sia destinato a cambiare spesso il proprio padre adottivo!

Figura 11.13 *Si può utilizzare l'API SetParent per spostare un controllo da un modulo a un altro.*

Vediamo come si impostano il pannello di controllo (che costituisce il form di avvio del progetto) e il caricamento degli altri tre moduli:

```
Option Explicit
Dim CurrentParent As Integer

Private Sub Form_Load()
   Form3.Show
   Form2.Show
   Form1.Show
   CurrentParent = 0 'Form1
   FormOnTop Me.hWnd, True
   Me.SetFocus
End Sub
```

Se non si chiama il metodo SetFocus(Me.SetFocus) per *il form del pannello di controllo, questo form non avrebbe il* focus, *cioè non sarebbe il form attivo, nel momento in cui si conclude la procedura di caricamento. Dato che tutta l'interazione prevista dal progetto ha luogo sul form del pannello di controllo, certamente voglio che abbia il focus.*

Cediamo l'evento di caricamento di Form1, che definisce lstBearsLike.

```
Private Sub Form_Load()
   lstBearsLike.AddItem "Honey"
   lstBearsLike.AddItem "Salmon Berries"
   lstBearsLike.AddItem "Salmon (the fish)"
   lstBearsLike.AddItem "Chocolate Covered Nuts"
End Sub
```

Si definisce True la proprietà .Sorted *di* lstBearsLike *in fase di progettazione in modo che la casella di riepilogo presenti le voci in ordine alfabetico.*

Vediamo la codifica della procedura relativa al clic cmdMove:

```
Private Sub cmdMove_Click()
   Dim Newform As Integer, I As Integer
   For I = 0 To 2
      If optForm(I) Then Newform = I
   Next I
   If Newform <> CurrentParent Then
      Select Case Newform
         Case 0 'Sposta su Form1
            MoveControl Form1.lstBearsLike, Form1
         Case 1 'Sposta su Form2
            MoveControl Form1.lstBearsLike, Form2
         Case 2 'Sposta su Form3
            MoveControl Form1.lstBearsLike, Form3
         Case Else
            MsgBox "Errore interno!"
         End Select
         CurrentParent = Newform
   End If
```

Questa routine richiama MoveControl con il form appropriato come parametro. Il programma dimostrativo a questo punto è pronto (si veda la Figura 11.13)!

È noto che la singola riga:

```
If optForm(I) Then Newform = I
```

è un'abbreviazione VB per:

```
If optForm(I).Value = True Then
   Newform = I
End If
```

Blocco degli utenti su un controllo

Supponiamo di volere che gli utenti di una applicazione debbano assolutamente fare qualcosa in un controllo prima di poterlo abbandonare. Per esempio, si consideri un riquadro che contiene una serie di pulsanti d'opzione. L'utente deve effettuare una scelta (facendo clic su uno dei pulsanti d'opzione) prima di poter fare qualunque altra cosa. Questo è in qualche modo equivalente al dialogo modale, nel quale l'utente deve fare clic su *OK* oppure su *Cancel* prima di proseguire. È facile rafforzare la modalità virtuale di un controllo vincolando il movimento del cursore

mediante le API. Se il cursore viene vincolato e non sono definiti accessi da testiera, l'utente può solo scegliere con il cursore (cioè, con il mouse) all'interno dell'area delimitata.

//progetto dimostrativo che si trova nel CD-ROM con il nome LockUsrs. Vbp blocca gli utenti all'interno di un riquadro. Un utente non può passare ad altro fino a quando non ha fatto clic su uno dei controlli nel riquadro.

È ovvio che si può definire la funzione ClipCursor per scopi diversi da quelli che riguardano un controllo "modale" (a scelta obbligatoria). Le dichiarazioni necessarie, ricavate da APIDec.Bas, sono le seguenti:

```
Declare Function GetDesktopWindow Lib "user32" () As Long
Declare Function GetWindowRect Lib "user32" _
   (ByVal hwnd As Long, lpRect As RECT) As Long
Declare Function ClipCursor Lib "user32" (lpRect As Any) As Long
Type RECT
   Left   As Long
   Top    As Long
   Right  As Long
   Bottom As Long
End Type
```

Se nella demo fraLock si aggiunge un riquadro che contiene una matrice di pulsanti d'opzione nel modulo d'avvio, è possibile bloccare i movimenti del cursore all'interno del riquadro aggiungendo il codice seguente in corrispondenza dell'evento di caricamento del modulo:

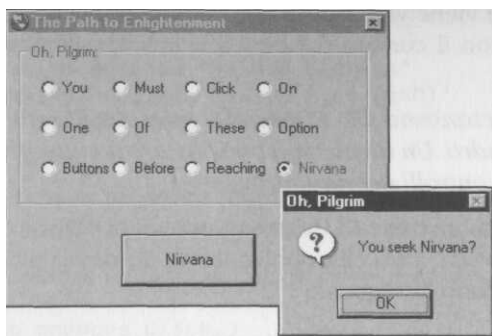
```
Private Sub Form_Load()
   GetWindowRect fraLock.hwnd, ControlRect
   ClipCursor ControlRect
End Sub
```

Ho sistemato il codice che sblocca il cursore nell'evento clic sulla matrice di pulsanti d'opzione, con il nome optLock (si veda il Listato 11.4).

Listato 11.4 *Utilizzo di ClipCursor per ripristinare il movimento del cursore.*

```
Private Sub optLock_Click(Index As Integer)
   Dim ScreenRect As RECT, ScreenHandle As Long
   ScreenHandle = GetDesktopWindow 'Prende l'handle dello schermo
   GetWindowRect ScreenHandle, ScreenRect
   ClipCursor ScreenRect 'Ripristina il cursore
End Sub
```

Possibile utilizzare la funzione GetDesktopWindow *per restituire un handle dell'intero schermo, come nell'esempio. Si può d'altro canto stabilire il libero accesso ali intero form; questo può essere fatto passando a* GetWindowRect *il gestore del forrn. In questo caso ci si deve assicurare a un ceno punto il ripristino dell'accesso allo schermo, di solito in corrispondenza dell'evento di scarico del form.*

Figura 11.14 *Sipuò utilizzare ClipCursor per assicurare che il cursore del mouse non si possa muovere all'esterno del riquadro che contiene un array dipulsanti d'opzione, fino a quando non ne viene premuto uno.*

Modifica del menu Window di una applicazione

Il menu *Window* compare quando si fa clic sulla piccola icona visibile nell'angolo superiore sinistro di un form; in Windows 3.x questo menu era conosciuto con il nome di menu di sistema o menu di controllo e compariva quando l'utente faceva clic sulla casella di controllo visibile nell'angolo superiore sinistro di un form. È possibile modificare il menu *Window* messo a disposizione dal sistema, per esempio rimuovendo la voce *Close* per essere sicuri che nessuno possa chiudere in questo modo il form. È facile modificare un form in questo senso; vediamo le dichiarazioni API necessarie, sempre ricavate da APIDec.Bas:

```
Declare Function GetSystemMenu Lib "user32" _
   (ByVal hwnd As Long, ByVal bRevert As Long) As Long
Declare Function RemoveMenu Lib "user32" _
   (ByVal hMenu As Long, ByVal nPosition As Long, _
   ByVal wFlags As Long) As Long
```

Il Listato 11.5 mostra la procedura generalizzata, che si trova in APICode.Bas, la quale cancella la voce *Close* (e la linea di separazione sopra *Close)* dal menu *Window* del form del quale riceve l'handle:

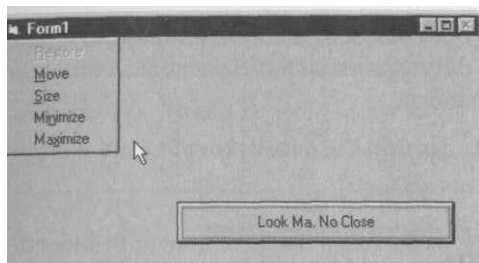
Listato 11.5 *Modifica del menu di sistema per evitare la chiusura.*

```
Public Sub TakeCloseOff(Handle As Long)
   Dim SysMenHandle As Long, RetVal As Long
   SysMenHandle = GetSystemMenu(Handle, 0)
    'Prende l'handle del menu di sistema di Form1
   RetVal = RemoveMenu(SysMenHandle, 6, MF_BYPOSITION)
    'Elimina la voce Close
   RetVal = RemoveMenu(SysMenHandle, 5, MF_BYPOSITION)
    'Elimina il separatore che ora si trova in basso
End Sub
```

La chiamata di questa procedura nel progetto dimostrativo (SysMenu.Vbp) è localizzata nell'evento clic del pulsante di comando cmdRemove:

```
TakeCloseOff cmdRemove.Parent.hwnd
```

Il risultato è un menu Window nel quale mancano la linea di separazione in basso e la voce *Close,* come mostrato in Figura 11.15-

Figura 11.15 *Si possono utilizzare le API per modificare il menu Window di un form.*

Non potrebbe essere più semplice! Vale la pena notare che la funzione RemoveMenu, *quando ha passato il/lag* MF_BYPOSITION, *cancella la voce del menu il cui numero è indicato dal penultimo parametro, contando da zero dall'alto verso il basso.*

Se si esegue il programma (o comunque si utilizza la procedura generalizzata) si può notare che Windows non solo rimuove la voce *Close* dal menu *Window,* ma disattiva anche il pulsante *Close* del modulo (la casella con la X nell'angolo superiore destro del modulo).

Controllo delle risorse minime di sistema

È possibile utilizzare TAPI GlobalMemoryStatus per stabilire la quantità delle diverse memorie disponibili su un sistema Windows a 32 bit.

Per stabilire le risorse disponibili si può anche utilizzare il controllo ActiveX Syslnfo, distribuito con la Professional Edition di VB6.

GlobalMemoryStatus richiede un argomento, una struttura di tipo MEMORYSTATUS. Questa struttura contiene otto variabili lunghe. La prima è un valore da passare a GlobalMemoryStatus che indica la quantità di memoria in byte che deve essere allecata per l'istanza della struttura MEMORYSTATUS. Dato che ciascuna delle otto variabili è dichiarata in VB come intero lungo, e che un intero lungo viene memorizzato in 4 byte, basta calcolare 4 * 8 = 32, e quindi passare 32 come primo elemento della struttura. Vediamo gli altri elementi, quelli restituiti da GlobalMemoryStatusCall e il loro significato:

- **dwMemoryLoad.** Un numero compreso tra 0 e 100 che fornisce un'idea generale dell'utilizzo corrente della memoria; 0 indica nessuna memoria utilizzata e 100 sta per memoria utilizzata completamente.
- **dwTotalPhys.** Numero totale di byte di memoria fisica presente.
- **dwAvailPhys.** Numero totale di byte di memoria fisica disponibile.

- **dwTotalPageFile.** Numero totale di byte che possono essere memorizzati nel file di paginazione; questo numero non rappresenta la dimensione corrente del file di paginazione su disco.
- **dwAvailPageFile.** Numero di byte effettivamente disponibili nel file di paginazione.
- **dwTotalVirtual.** Numero totale di byte della Forzione in modalità utente dello spazio di indirizzo virtuale relativa al processo chiamante.
- **dwAvailVirtual.** Numero di byte della memoria a disposizione nella Forzione in modalità utente dello spazio di indirizzamento virtuale, relativa al processo che si sta chiamando.

Vediamo la dichiarazione della funzione GlobalMemoryStatus e il tipo MEMORYSTATUS, da APIDec.Bas:

```
Declare Sub GlobalMemoryStatus Lib "kernel32" _
   (lpBuffer As MEMORYSTATUS)
Type MEMORYSTATUS
   dwLength As Long
   dwMemoryLoad As Long
   dwTotalPhys As Long
   dwAvailPhys As Long
   dwTotalPageFile As Long
   dwAvailPageFile As Long
   dwTotalVirtual As Long
   dwAvailVirtual As Long
End Type
```

La procedura generalizzata che richiama GlobalMemoryStatus si trova nel modulo APICode.Bas con il nome CheckResources. La funzione CheckResources ripassa indietro come parametri i diversi valori degli elementi presenti nella struttura MEMORYSTATUS. Inoltre, accetta l'argomento BlowAlarm che definisce una percentuale al di sotto della quale viene attivato un allarme di sistema. BlowAlarm viene confrontato con il valore 100 meno l'elemento .dwMemoryLoad; se BlowAlarm è inferiore a questo valore calcolato, CheckResources restituisce True, altrimenti False. Questo consente di scegliere azioni appropriate in funzione della situazione, come l'invio di un messaggio di avvertimento oppure il mancato caricamento di librerie addizionali. Il Listato 11.6 contiene la funzione CheckResources:

Listato 11.6 *Controllo delle risorse di sistema.*

```
Public Function CheckResources(Percent As Long, _
   Optional BytesPhys As Variant, Optional FreePhys As Variant, _
   Optional BytePage As Variant, Optional FreePage As Variant, _
   Optional UserBytes As Variant, Optional FreeUser As Variant, _
      Optional BlowAlarm As Variant) As Boolean
   Dim HowMuchMemory As MEMORYSTATUS
   HowMuchMemory.dwLength = 32
   GlobalMemoryStatus  HowMuchMemory
   Percent = HowMuchMemory.dwMemoryLoad
   BytesPhys = HowMuchMemory.dwTotalPhys
```

```
    FreePhys = HowMuchMemory.dwAvailPhys
    BytePage = HowMuchMemory.dwTotalPageFile
    FreePage = HowMuchMemory.dwAvailPageFile
    UserBytes = HowMuchMemory.dwTotalVirtual
    FreeUser = HowMuchMemory.dwAvailVirtual
    If Not IsMissing(BlowAlarm) Then
       If BlowAlarm >= 100 - Percent Then
          CheckResources = False 'Scatta l'allarme
       Else
          CheckResources = True
       End If
    Else
       CheckResources = True
    End If
End Function
```

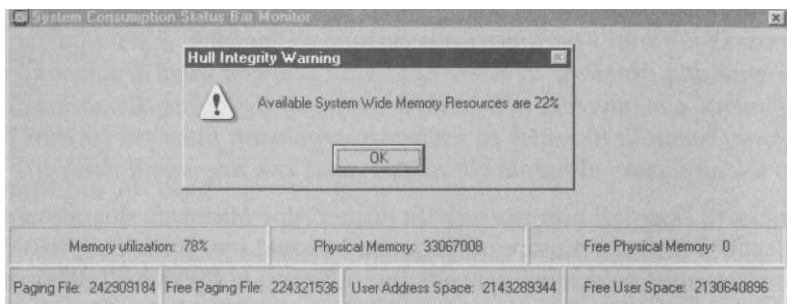
La funzione CheckResources *è stata dichiarata utilizzando la parola chiave* Optional *per tutti i parametri a eccezione di* Percent, *il primo della lista. Questo consente alla funzione di essere chiamata solo con quell'argomento, o quei pochi argomenti, che interessano (si veda per esempio il progetto About Box della prossima sezione) senza la necessità di utilizzare argomenti fittizi per gli altri parametri. Si può solo utilizzare la parola chiave* Optional *con argomenti variant.*

Per essere sicuro di non ricevere un errore Type Mismatch quando si chiama CheckResources senza passare un parametro BlowAlarm, come nel prossimo esempio, in cui interessa solamente Percent, ho utilizzato la funzione IsMissing per verificare la presenza di BlowAlarm. Se questo manca, si salta il confronto che altrimenti genererebbe un errore.
Nel progetto LowRes.Vbp, che illustra l'utilizzo di questa funzione, sono state aggiunte due barre di stato in basso nel forni (si veda la Figura 11.16). Le barre di stato sono continuamente aggiornate con informazioni sul sistema da un controllo Timer che si attiva ogni tre secondi. Come mostrato nel Listato 11.7, la codifica dell'evento Timer di Timer richiama la funzione CheckResources che aggiorna la barra di stato; se la memoria complessiva è inferiore al 25% viene visualizzato un messaggio di avvertimento.

Listato 11.7 *Visualizzazione dello stato del sistema.*

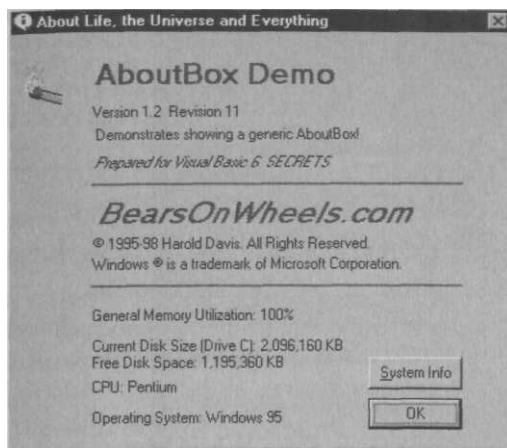
```
Private Sub Timer1_Timer()
   Dim x As Boolean, Percent As Long, BytesPhys As Long, _
      FreePhys As Long, BytePage As Long, FreePage As Long, _
      UserBytes As Long, FreeUser As Long, BlowAlarm As Integer
   BlowAlarm = 25 'Blow alarm if general memory less than!
   x = CheckResources(Percent, BytesPhys, FreePhys, BytePage, _
      FreePage, UserBytes, FreeUser, BlowAlarm)
   sbSystem1.Visible = True
    bSystem2.Visible = True
   sbSystem1.Panels(1).Text = "Memory utilization: " + _
      Str(Percent) + "%"
   sbSystem1 .Panels(2).Text = "Physical Memory: " + _
```

```
    Str(BytesPhys)
  sbSystem1.Panels(3).Text = "Free Physical Memory: " _
    + Str(FreePhys)
  sbSystem2.Panels(1).Text = "Paging File: " + Str(BytePage)
  sbSystem2.Panels(2).Text = "Free Paging File: " + _
    Str(FreePage)
  sbSystem2.Panels(3).Text = "User Address Space: " _
    + Str(UserBytes)
  sbSystem2.Panels(4).Text = "Free User Space: " + Str(FreeUser)
  If Not x Then
    MsgBox "Available System Wide Memory Resources are" + _
      Str(100 - Percent) +
      "%", vbExclamation, "Hull Integrity Warning"
  End If
End Sub
```

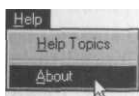
Figura 11.16 *Si può utilizzare GlobalMemory-Status per controllare le informazioni sul sistema e per attivare un allarme nel caso in cui le risorse vadano al di sotto di una determinata percentuale.*

A questo proposito, se si compila il programma ci si accorge di avere tra le mani un semplice sistema per controllare le risorse in uso. Se non vi piace così com'è, potete formattare diversamente le stringhe di visualizzazione per renderle più leggibili. Il programma può controllare il consumo incrementale di risorse da parte delle applicazioni a partire dal momento in cui vengono aperte.

Una casella About per visualizzare informazioni sul sistema

In questa sezione vediamo come si crea una casella About *generica e riusabile; il progetto si trova nel CD-ROM allegato al libro con il nome di About. Vbp. La parte inferiore della casella* About *visualizza informazioni sul sistema, come mostrato in Figura 11.17; vedremo come si ottengono e si visualizzano questi dati (vedremo anche come richiamare Microsoft System Information Utility). La pane superiore della casella* About *visualizza informazioni che riguardano il fornitore dell'applicazione e del software. L'aspetto più interessante riguarda il fatto che, ad eccezione dell'icona, tutte queste informazioni vengono caricate automaticamente dalle proprietà* App *dell'oggetto. Queste proprietà sono visibili nella scheda* Make *del dialogo* Project Properties *(Figura 11.19).*

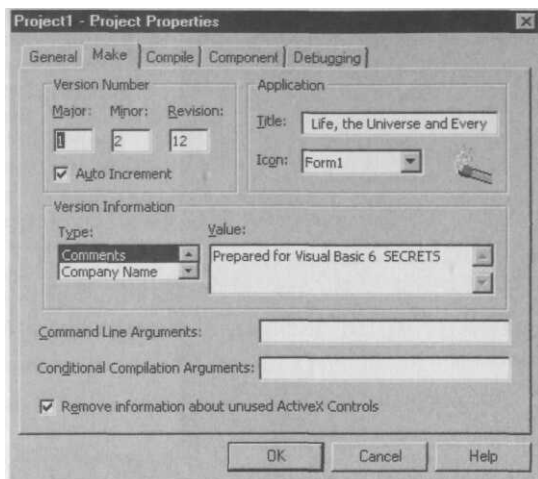
Figura 11.17 *L'oggetto App e le API di Windows consentono di creare facilmente una casella About generica. riutilizzabile e personalizzabile in modo automatico, che includa informazioni sul sistema.*

È possibile accedere alle impostazioni relative alle opzioni *Make* mediante *Make* nel menu *File* oppure selezionando *Properties* nel menu *Project.* Non occorre mai intervenire manualmente sulla casella *About;* basta inserire nel progetto questa e i moduli con le dichiarazioni e le procedure di supForto.

Prima le cose da fare innanzi tutto. Il form della casella *About,* di nome AboutBox, viene caricato da mnuAbout nel menu *Help* (si veda la Figura 11.18). Questo menu è organizzato nella consueta forma di Windows, con una singola voce *Help Topics* seguita da una linea di separazione, seguita a sua volta dalla voce *About.*

Figura 11.18 *La voce About inserita nel menu Help.*

Come al solito, per caricare la casella *About* si utilizza il suo metodo Show, anche se spesso le caselle *About* sono di tipo modale.

```
Private Sub mnuAbout_Click()
   AboutBox.Show vbModal
End Sub
```

Ho utilizzato il metodo Show di AboutBox con la costante vbModal come argomento; in questo modo si rende AboutBox *modale.* La casella deve cioè essere chiusa prima che l'utente possa fare qualunque altra cosa al di fuori di questa .VbModal (oppure 1)e VbModeless (0) sono costanti VB6; per definizione, se non viene indicata la costante nel metodo Show, ci si trova con una casella non modale. I controlli di AboutBox (molte etichette e una immagine) sono in effetti stabilite dalla procedura di caricamento di AboutBox. Una prima parte di questi utilizza come valori alcune proprietà dell'oggetto App. (Per un elenco completo delle proprietà dell'oggetto App, si consulti la guida online di VB6.) Per impostare le proprietà dell'oggetto App relative a un particolare progetto, si utilizza la scheda *Make* del dialogo *Project Properties* (si veda la Figura 11.19), quindi si compila il progetto.

Figura 11.19 *La scheda Make el dialogo Project Properties consente di modificare le proprietà dell'oggetto App del progetto.*

Una nuova caratteristica detta scheda Make *del dialogo* Project Properties *riguarda la casella di spunta* Remove Information About Unused ActiveX Controls *(si veda la Figura 11.19). Se questa casella è attiva, il compilatore rimuove automaticamente le informazioni relative ai controlli dei quali si fa riferimento nel progetto anche se non vengono utilizzati. Questa opzione può risultare utile in quanto i controlli vengono spessi inseriti nel progetto in fase di definizione anche se, successivamente, si decide di non utilizzarli. Questa opzione rende automatica la procedura di rimozione del riferimento a questi controlli, Fortando a un eseguibile che utilizza meno risorse.*

Il Listato 11.8 mostra la prima parte del codice dell'evento di caricamento di AboutBox. Tutte le etichette collegate all'applicazione sono riempite in modo automatico sulla base dei valori relativi alla proprietà App. L'unica modifica richiesta per utilizzare questo codice in una applicazione riguarda la posizione dell'icona che deve essere caricata in imgAppIcon (presumendo che questa posizione non sia Form1).

Listato 11.8 *Visualizzazione di informazioni sull'applicazione.*

```
Private Sub Form_Load()
   Dim Percent As Long, x As Boolean, DiskSize, FreeKB, _
     Drive As String, dw As Long, ThisOs As String, _
     CPU As String, ThisSys As SYSTEM_INFO
   Me.Caption = "About " + App.Title
    'Centra le AboutBox sullo schermo
   CenterForm Me
    'La prossima è l'unica riga che dovete cambiare
   imgAppIcon.Picture = Form1.Icon

   lblAppName.Caption = App.ProductName
   lblVersion.Caption = "Version " & App.Major & "." & _
     App.Minor & "  Revision " & App.Revision
   lblDescription.Caption  = App.FileDescription
```

```
lblComments.Caption = App.Comments
lblCompany.Caption = App.CompanyName
lblCopyright.Caption = Chr(169) + " " + App.LegalCopyright & _
   " Ali Rights Reserved."
lblTrademark.Caption = App.LegalTrademarks
```

Si può utilizzare Chr(l69)per avere un simbolo di copyright alposto di (C); il codice ASCIl corrispondente al simbolo di copyright è infatti 169.

Il resto della procedura di caricamento sistema le informazioni nelle etichette che riguardano l'utilizzo della memoria, lo spazio su disco, la CPU e il sistema operativo, come mostrato in Figura 11.9.

Listato 11.9 *Visualizzazione di informazioni sul sistema.*

```
 ' Prende informazioni sulla configurazione di Windows e sul sistema
 ' Gestisce solo sistemi operativi a 32 bit;
 ' Usa la compilazione condizionale per trattare i 16 bit
x = CheckResources(Percent)
lblResources.Caption = "General Memory Utilization: " & _
   Percent & "%"
GetDisk DiskSize, FreeKB, Drive
lblDiskSpace.Caption = "Current Disk Size (Drive " + _
   Left(Drive, 1)+ "): " + Format(DiskSize, "###,###") +_
   " KB" + vbCrLf + " Free Disk Space: " + _
   Format(FreeKB, "###,###") + " KB"
dw = GetVersion()
If dw And &H80000000 Then
   ThisOs = "Windows 95/98"
Else
   ThisOs = "Windows NT"
End If
lblOS.Caption = "Operating System: " + ThisOs
GetSystemlnfo ThisSys
Select Case ThisSys.dwProcessorType
   Case 386
      CPU = "386"
   Case 486
      CPU = "486"
   Case 586
      CPU = "Pentium"
   Case 2000
      CPU = "R2000"
   Case 3000
      CPU = "R3000"
   Case 4000
      CPU = "R4000"
   Case 21064
      CPU = "A21064"
   Case Else
      CPU = "Processore sconosciuto!"
```

```
   End Select
   lblCPU.Caption = "CPU: " + CPU
End Sub
```

In primo luogo, si determina l'utilizzo della memoria generale mediante il primo argomento restituito dalla funzione CheckResources, che si trova nel modulo APICode.Bas. Abbiamo già visto in questo capitolo come funziona CheckResources e al sua impostazione mediante la parola chiave Optional e la funzione IsMissing in modo che la si possa chiamare con il suo primo parametro soltanto. Poi si chiama la procedura GetDisk, la quale trova il nome del disco corrente e riForta come parametri lo spazio totale sul disco, lo spazio disponibile sul disco e il nome del disco. Il Listato 10.11 mostra la codifica di GetDisk, ricavata da APICode.Bas:

Listato 11.10 *Ricerca dello spazio disponibile su disco.*

```
Public Sub GetDisk(DiskSize, FreeKB, Drive As String)
   Dim x As Boolean, SectorsPerCluster As Long, _
      BytesPerSector As Long, FreeClusters As Long, _
         TotalClusters As Long,Buffer As String
   Buffer = Space(255)
   x = GetCurrentDirectory(Len(Buffer), Buffer)
   Drive = Left(ConvertCToVBString(Buffer), 3)
   x = GetDiskFreeSpace(Drive, SectorsPerCluster, _
         BytesPerSector, FreeClusters, TotalClusters)
   If x Then
      DiskSize = (SectorsPerCluster * BytesPerSector * _
         TotalClusters) \ 1024 'Convert to KB
      FreeKB = (SectorsPerCluster * BytesPerSector * _
         FreeClusters) \ 1024 'Convert to
   Else
      MsgBox "Errore interno di GetDisk!"
   End If
End Sub
```

Il passo successivo riguarda la conversione della stringa in stile C restituita da GetCurrentDirectory in una stringa VB; si deve tagliare l'ultimo carattere, il terminatore nullo. Dato che probabilmente questa operazione andrà ripetuta spesso, l'ho inserita nella funzione del Listato 11.11 (sempre in APICode.Bas):

Listato 11.11 *Conversione di una strìnga in stile C in una strìnga VB.*

```
PublicFunctionConvertCToVBString(InStringAsString)AsString
   ConvertCToVBString = Left(InString, Len(InString) - 1)
End Function
```

Per individuare la denominazione del disco (che è la cosa a cui siamo veramente interessati) è sufficiente controllare i primi tre caratteri della directory corrente:

```
Drive = Left(ConvertCToVBString(Buffer), 3)
```

infine si richiama GetFreeDiskSpace, passando il suo Drive come primo parametro; la dichiarazione di GetFreeDiskSpace è proprio quella che ci si aspetta di trovare e si trova in APICode.Bas. Il passo successivo consiste nella lettura dei valori di ritorno della funzione, da convertire in chilobyte, prima che vengano riportati.

Si ritorna all'evento di caricamento AboutBox per formattare le informazioni relative alloo spazio su disco e per aggiungerle alle etichette del modulo.

La funzione Format *semplifica tutto quello che riguarda la formattazione; per esempio,*

```
Format(DiskSize, "###,###"),
```

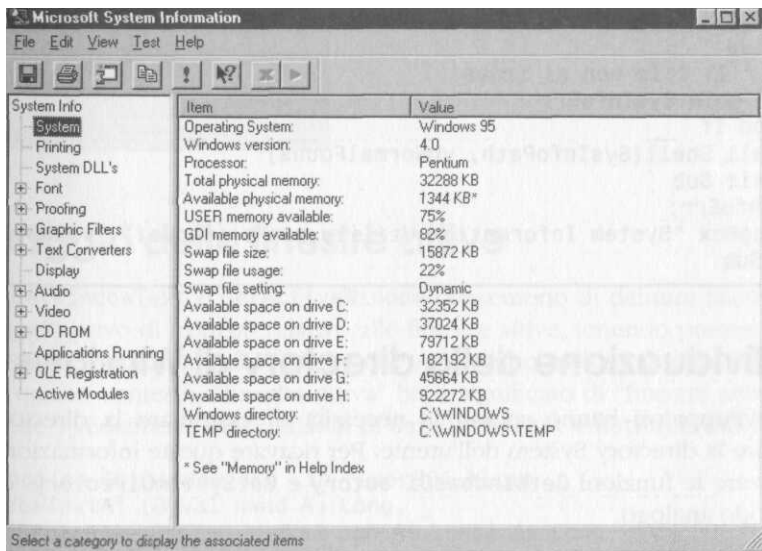
fa sì che vengano inserite delle virgole ogni tre cifre di un numero a sei cifre (o meno).

Il passo conclusivo consiste nello stabilire le informazioni su CPU e sistema operativo da inserire nel form About mediante due API: GetVersion e GetSystemInfo.

Si può notare che VB6 prevede un dialogo About *che si può utilizzare come modello; è sufficiente aggiungere un form al proprio progetto e selezionare* About Dialog *nella scheda* New *del dialogo* Add Form. *Questo dialogo About precostituito utilizza le proprietà dell'oggetto* App, *in modo analogo alla casella About personalizzata di questa sezione, anche se l'aspetto dei due dialoghi presenta delle differenze.*

Microsoft System Information Utility

Il dialogo *About* di VB6 e quello personalizzato in questa sezione consentono entrambi di richiamare Microsoft System Information Utility (si veda la Figura 11.20) quando si fa clic su un pulsante di comando.

Figura 11.20 *La vostra applicazione può consentire all'utente di richiamare Microsoft System Information Utility, la quale visualizza una serie di informazioni che riguardano il sistema dell'utente.*

Le dichiarazioni per le funzioni API, le costanti e tutto il codice necessario sono nel CD-ROM allegato al libro nel modulo SysInfo.Bas. La procedura principale, StartSysInfo, prova a stabilire e confermare posizione ed esistenza dell'eseguibile relativo a System Information (Msinfo32.Exe). Se non viene trovato, lo si lancia utilizzando la funzione Shell. *// modulo definisce molte costanti, incluse quelle richieste per individuare Msinfo32.Exe nel registro:*

```
Const gREGKEYSYSINFOLOC = "SOFTWARE\Microsoft\Shared Tools Location"
Const gREGVALSYSINFOLOC = "MSINFO"
Const gREGKEYSYSINFO = "SOFTWARE\Microsoft\Shared Tools\MSINFO"
Const gREGVALSYSINFO = "PATH"
```

Il Listato 11.12 mostra il codice corrispondente:

Listato 11.12 *Avvio dell'utility Microsoft System Information.*

```
Public Sub StartSysInfo()
   On Error GoTo SysInfoErr
   Dim rc As Long
   Dim SysInfoPath As String
   If GetKeyValue(HKEY_LOCAL_MACHINE, _
      gREGKEYSYSINFO, gREGVALSYSINFO, SysInfoPath) Then
   ElseIf GetKeyValue(HKEY_LOCAL_MACHINE, _
      gREGKEYSYSINFOLOC, gREGVALSYSINFOLOC, _
      SysInfoPath) Then
      If (Dir(SysInfoPath & "\Msinfo32.Exe") <> "") Then
         SysInfoPath = SysInfoPath & "\Msinfo32.Exe
      Else
         GoTo SysInfoErr
      End If
   Else
      'Il file non si trova!
      GoTo SysInfoErr
   End If
   Call Shell(SysInfoPath, vbNormalFocus)
   Exit Sub
SysInfoErr:
   MsgBox "System Information Utility non si trova!", vbOKOnly
End Sub
```

Individuazione della directory di Windows

Gli sviluppatori hanno spesso la necessità di esaminare la directory Windows oppure la directory System dell'utente. Per ricavare queste informazioni si possono utilizzare le funzioni GetWindowsDirectory e GetSystemDirectory, che lavorano in modo analogo.

Vale la pena notare che non conviene mai copiare file nella directory Windows\System perché, se l'utente sta lavorando con una versione condivisa di Windows perché l'applicazione potrebbe non avere accesso in scrittura a quella directory. La directory restituita da GetWindowsDirectory *ha lo scopo di risolvere la questione.*

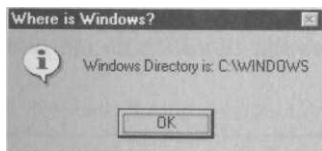
In generale conviene al contrario aggiungere quanti più file possibile nelle posizioni "pubbliche". La situazione migliore è quella che vede tutti i file del vostro programma nella directory e nella struttura di file che l'installazione crea appositamente.

Vediamo la dichiarazione relativa a GetWindowsDirectory:

```
Declare Function GetWindowsDirectory Lib "kernel32" Alias _
   "GetWindowsDirectoryA" (ByVal lpBuffer As String, _
   ByVal nSize As Long) As Long
```

Si può utilizzare il codice seguente per visualizzare la posizione della directory Windows in un'apposita casella, come mostrato in Figura 11.21:

```
Private Sub Form_Click()
   Dim Buffer As String, x As Long
   Buffer = Space(255)
   x = GetWindowsDirectory(Buffer, Len(Buffer))
   MsgBox "Windows Directory is: " + _
      ConvertCToVBString(Buffer), vbInformation, _
      "Dov'è Windows?"
End Sub
```

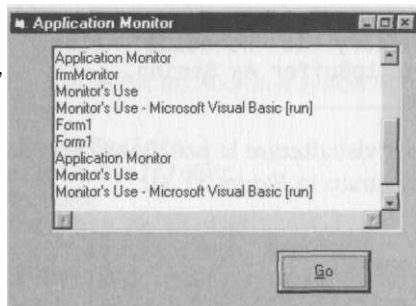
Figura 11.21 *La funzione GetWindows-Directory da la posizione*

Monitoraggio delle finestre attive

Le funzioni GetWindowText e GetActiveWindow consentono di definire facilmente un monitoraggio privo di fronzoli relativo alle finestre attive, tenendo presente che queste funzioni restituiscono informazioni che riguardano esclusivamente il thread corrente; in questo contesto, "finestra attiva" ha il significato di "finestra attiva nel thread corrente". Vediamo le dichiarazioni di GetWindowText e GetActiveWindow:

```
Declare Function GetWindowText Lib "user32" Alias _
   "GetWindowTextA" (ByVal hwnd As Long, _
   ByVal lpString As String, ByVal cch As Long) As Long
Declare Function GetActiveWindow Lib "user32" () As Long
```

Vediamo il codice, posto in un evento Timer, che richiama queste funzioni e sistema i risultati in un controllo a casella di testo multiriga ogni volta che scatta il Timer (si veda la Figura 11.22):

```
Private Sub Timer1_Timer()
   Dim hCurrent As Long, Buffer As String, nChars As Integer
   Buffer = Space(255)
   hCurrent = GetActiveWindow()
   nChars = GetWindowText(hCurrent, Buffer, 255)
   txtMonitor.Text = txtMonitor.Text & vbCrLf & _
      ConvertCToVBString(Buffer)
End Sub
```

Figura 11.22
Si utilizzano SetActiveWindow e GetActiveText per rilevare il contenuto del titolo di una finestra attiva.

Si può utilizzare il metodo Print *dell'oggetto* Debug *per visualizzare i valori durante l'esecuzione di un programma VB in ambiente di sviluppo. Per esempio:*

```
Debug.Print ConvertCToVBString(Buffer)
```

inserito in aggiunta alla casella di testo dell'esempio precedente (oppure al suo posto) fa sì che il testo della finestra attiva venga visualizzato nel pannello Immediate *di VB.*

Per andare oltre

I progetti dimostrativi di questa sezione hanno illustrato alcuni impieghi delle API che consentono di estendere le possibilità di Visual Basic (date anche un'occhiata più in avanti alla sezione che si occupa del sistema di messaggi di Windows). E ovvio che ci sarebbe ancora molto da dire! Gli esempi trattati non fanno altro che intaccare la superficie dell'intera questione. Non è sempre facile stabilire come usare da VB le funzioni API, ma se ci si riesce, la ricompensa è un numero elevatissimo di funzioni.

Inoltre, molte funzioni fanno la stessa cosa dei corrispondenti controlli personalizzati. Per esempio, è possibile utilizzare GetOpenFileName in Comflg32.Dll per aprire un dialogo comune *Open* senza avere a che fare con il controllo dei dialoghi comuni (si veda il Capitolo 7 per maggiori informazioni su Comdlg32.Ocx). Insomma, non ci sono limiti a quello che si può fare una volta che si conoscono i fondamenti delle API.

Determinazione del sistema operativo

La funzione API GetVersionEx *consente di ottenere facilmente informazioni che riguardano il sistema operativo con il quale si sta lavorando. Le dichiarazioni, le costanti e il codice che servono per sapere con quale sistema operativo sta lavorando il vostro programma sono memorizzati nel CD-ROM allegato al libro nel modulo WhichOS.Bas relativo al progetto WhichOS.Vbp. Il Listato 11.3 contiene costanti, tipi e dichiarazioni API del modulo:*

Listato 11.13 *Dichiarazioni per l'utilizzo di GetVersionEx.*

```
Public Const VER_PLATFORM_WIN32_NT = 2
Public Const VER_PLATFORM_WIN32_WINDOWS = 1
Public Const VER_PLATFORM_WIN32s = 0

Type OSVERSIONINFO
   dwOSVersionInfoSize As Long
   dwMajorVersion As Long
   dwMinorVersion As Long
   dwBuildNumber As Long
   dwPlatformId As Long
   szCSDVersion As String * 128        '  Stringa di manutenzione
End Type
Declare Function GetVersionEx Lib "kerne!32"
   Alias "GetVersionExA" (lpVersionInformation As _
   OSVERSIONINFO) As Long
```

Il Listato 11.4 mostra il codice, relativo alla procedura Sub Main, che conclude il programma se non si sta lavorando con Windows NT oppure con Windows 95/98, altrimenti visualizza informazioni sul sistema operativo.

Listato 11.14 *Determinazione del sistema operativo.*

```
Public Sub Main()
   Dim OS As OSVERSIONINFO
   OS.dwOSVersionInfoSize = Len(OS)
   GetVersionEx OS
   If OS.dwMajorVersion < 4 Then
      MsgBox "Mi dispiace, servono Windows 95/98 o NT4 o sucessivi!", _
         vbInformation, "Questo programma è andato!"
      End 'Termina l'esecuzione
   Else
      Debug.Print "OK. Ci siamo! Windows 95/98 o NT4..."
      Select Case OS.dwPlatformId
         Case VER_PLATFORM_WIN32_WINDOWS
            MsgBox "Win 32 e Windows 95/98 in esecuzione!", _
               vbInformation, "Visual Basic 6 Secrets"
         Case VER_PLATFORM_WIN32_NT
            MsgBox "Windows NT Version" & _
               Str(OS.dwMajorVersion) &
               " Build " & Str(OS.dwBuildNumber) & " in esecuzione!", _
               vbInformation, "Visual Basic 6 Secrets"
```

```
        Case Else
            MsgBox "Non ho indizi!", _
              vbInformation, "Visual Basic 6 Secrets"
        End Select
    End If
End Sub
```

Problemi comuni

Alcuni fra i problemi più comuni sono provocati dalle differenze tra Windows 95/98 e NT, in particolare:

- Diverso trattamento delle stringhe con terminatore nullo.
- Codifica rigida di locazioni del registro che non coincidono nei due sistemi operativi
- Codifica rigida di posizioni dei file che non coincidono nei due sistemi operativi.
- Implementazione più completa dei criteri di protezione e dello schema dei diritti di accesso in Windows NT.
- Diverso trattamento delle chiamate di handle a 16 bit tradizionali.
- Diverso trattamento dell'accesso a componenti hardware tra Windows 957 98 e NT.

Stringhe con terminatore nullo

Una stringa con terminatore nullo, o in stile C, utilizza un carattere nullo (un carattere ASCII di valore 0) per segnalare la fine di una stringa. Se si vuole utilizzare in VB una stringa con terminatore nullo, bisogna prima tagliare via Chr(O) dalla fine della stringa.
Alcuni valori del Registro sono memorizzati come stringhe con terminatore nullo in Windows 95/98 e senza terminatore in Windows NT (si veda il Capitolo 10 per maggiori informazioni). Non è difficile usare una funzione che elimina l'eventuale terminatore nullo presente, altrimenti lascia la stringa intatta, come succede nel Listato 11.15:

Listato 11.15 *Eliminazione del terminatore nullo, se esiste.*

```
Public Function ConvertString(tmpVal As String, _
    KeyValSize As Long) As String
    If (Asc(Mid(tmpVal, KeyValSize, 1)) =0) Then
        'Stringa Win95 con terminatore: eliminalo
        ConvertString = Left(tmpVal, KeyValSize - 1)
    Else
        'Windows NT non ha terminatone a fine stringa
        ConvertString = Left(tmpVal, KeyValSize)
        'Null non trovato, restituisce solo la stringa
    End If
EndFunction
```

Codifica rigida

La risposta ai problemi che sorgono quando un programmatore assume che una struttura del Registro di configurazione o un file si trovino in una posizione specifica, ed effettua di conseguenza una codifica rigida, è semplice: non fate mai una cosa simile. Si deve ricordare che la struttura del Registro è differente in Windows 95/98 rispetto a Windows NT, così come sono diverse le posizioni di imFortanti file di sistema. Controllate la posizione della struttura o di una chiave del registro, oppure la locazione di un file particolare (a meno che non abbiate sistemato voi stessi la voce o il file, ma anche in questo caso non è una brutta idea fare una verifica); il Capitolo 13 mette a disposizione le tecniche che consentono di referire un file su un sistema, se non si trova dove ci si aspetta che sia.

Le chiamate a Winhelp.Exe con codifica rigida presentano un problema particolare. Se si richiama WinHelp con un file di guida versione 2.0 come suo argomento, questo funziona con Windows 95/98 mentre fallisce (con un messaggio di errore) con Windows NT4. La procedura corretta è creare un processo con il file di guida, la cui estensione .Hlp farà automaticamente partire l'appropriato motore di guida su entrambi i sistemi.

Sicurezza e accessi

Windows NT è stato progettato tenendo ben presenti i problemi legati alla sicurezza. Un'applicazione che si crea eredita diritti, autorizzazioni e limitazioni dell'utente che esegue il programma con Windows NT. Questo significa che la vostra applicazione può non essere in grado di memorizzare file oppure di scrivere nel Registro (a meno che l'utente stia lavorando con privilegi da amministratore).

Handle a 16 bit

Un handle rappresenta un modo per accedere a una risorsa di Windows, come una finestra. Windows 3.x utilizzava handle a 16 bit; API Win32 utilizza handle a 32 bit. Nelle versioni a 32 bit di VB, gli handle sono memorizzati come variabili Long; nei VB a 16 bit erano memorizzati come Integer. Quando Windows 95/98 incontra un handle a 16 bit, lo completa con una serie di zeri e lo converte automaticamente in un handle a 32 bit. Windows NT non fa niente di tutto questo, per cui il vecchio codice che utilizza handle a 16 bit può funzionare con successo con Windows 957 98, ma fallisce sicuramente con NT 4. La risposta a questo problema consiste ovviamente nell'assicurarsi che il proprio vecchio codice sia stato completamente riscritto e che non contenga handle a 16 bit prima di utilizzarlo con sistemi operativi a 32 bit.

ANSI e Unicode

Unicode è un sistema di codifica dei caratteri adeguato per quasi tutte le lingue scritte del mondo: contiene per esempio i set di caratteri cirillico, greco, romano, tailandese, turco, arabo, ebraico e giapponese Kana. ANSI è un sistema di codifica dei caratteri molto più limitato, che richiede però una minori risorse di memoria,

basato sul set di caratteri latini. ANSI è il set di caratteri nativo per le stringhe di Windows 95/98, mentre Windows NT utilizza esclusivamente Unicode per la gestione interna di stringhe.
Le API Win32 sono state progettate in modo che ciascuna funzione inclusa preveda due forme: una valida per Unicode, l'altra per ANSI. Le due funzioni hanno un unico nome; è compito del compilatore stabilire quale forma deve essere richiamata.
Anche se Windows NT svolge il suo lavoro con Unicode, non si aspetta necessariamente che gli si parli in Unicode. Una funzione API chiamata in Windows NT con una stringa ANSI chiama un'altra API, MultiByteToWideChar, la quale converte la stringa ANSI in Unicode. Viene quindi eseguita la versione Unicode della API. Le stringhe sono poi convenite da Unicode a ANSI per mezzo di un'altra API, WideCharToMultiByte.
Le stringhe Unicode sono elaborate internamente in modo più efficiente, così come sono in grado di comprendere i caratteri del mondo intero (o quasi). Non si deve preparare codice diverso in VB a seconda del due diversi sistemi di codifica, ma in un futuro prossimo il supForto ANSI verrà molto probabilmente abbandonato. È quindi tempo di cominciare a pensare in termini di Unicode.

Utilizzo delle API Win32s

Le API Win32s sono un sottoinsieme delle API Win32. Il loro scopo è quello di consentire la creazione di applicazioni a 16 bit per Windows 3.1 che siano equivalenti a quelle per piattaforme a 32 bit (Windows 95/98 e NT). Questo risultato è possibile a patto che:

- Si resti aderenti alle funzioni incluse nel sottoinsieme Win32s.
- Si includa un codice condizionale in modo che il codice in fase di esecuzione dipenda dal sistema operativo.
- Si distribuiscano i programmi a 16 bit con le DLL che consentano ai programmi Win32s di lavorare con Windows 3.x.
- Ci si assicuri che i file Win32s siano installati su tutti i sistemi a 16 bit da utilizzare.

Win32s SDK viene fornito con un programma di installazione che rende automatica la procedura che carica le estensioni della piattaforma sui sistemi a 16 bit. Le applicazioni Win32s possono essere create solo utilizzando piattaforme a 32 bit (Windows 95/98 e NT). Si può contattare Microsoft per sapere come ottenere Win32s SDK, dato che non viene distribuito con Visual Studio 6.
Il set di funzioni supFortato da Win32s è sorprendentemente ricco. Ci si aspetta che le prossime versioni supFortino i controlli dell'interfaccia utente di Windows 95/98 anche se, ovviamente, alcune caratteristiche di Windows 95/98, come il multithreading, non potranno essere mai simulate in Windows 3.x. Per avere maggiori dettagli su cosa è disponibile in Wind32s si consulti Win32s SDK.
Dato che l'intera API Win32 viene esFortata da Win32s, ogni applicazione basata su Win32 può essere caricata in Windows 3.x; tuttavia, questo non significa che ogni API Win32 sia in grado di funzionare correttamente. Anche se tutte le API Win32

vengono esportate da Win32s, le funzioni Win32 che non possono essere implementate in Windows 3.x (come i percorsi, i thread, le trasformazioni e FI/O su file asincrono) sono destinate a fallire ed a riFortare codici di errore.
I codici di errore restituiti dalle API a 32 bit che non sono supFortate in Win32s dipendono dalle funzioni API. Di solito Win32s imposta il codice dell'ultimo errore a ERROR_CALL_NOT_IMPLEMENTED, che si può riprendere utilizzando l'API GetLastError (implementata in Win 32s). Tuttavia, le applicazioni basate su Win32 non devono affidarsi esclusivamente ai codici di errore, ma possono determinare in anticipo quale piattaforma Windows è in esecuzione richiamando la funzione GetVersionEx.
È ovvio che la cosa migliore è non sviluppare applicazioni che vengano eseguite a 16 bit (sarebbe anzi ora di guardare avanti verso quelle a 64 bit) anche se può essere necessario impostare la propria applicazione in modo che 'preveda, sulla base di una compilazione condizionale, una funzione per implementare caratteristiche differenti a seconda che lavori con Windows 3.1 e Win32s oppure con Windows a 32 bit. L'ultima versione di Visual Basic che supFortava lo sviluppo a 16 bit è stata VB4; l'ultima versione di Visual C++ è stata la 4.2.

Il sistema di messaggi di Windows

L'idea che sta alla base del sistema di messaggi di Windows è che Windows notifica gli eventi alle applicazioni inviando loro dei messaggi. Immaginiamo un flusso regolare di messaggi, per esempio ogni volta che si sposta il mouse. Alcuni, forse molti di quei messaggi sono imFortanti per la vostra applicazione. Mentre la vostra applicazione elabora i suoi messaggi, si trova che alcuni messaggi sono rilevanti (quindi, devono essere elaborati da) una determinata "finestra" (un forni o un controllo a finestra) dell'applicazione stessa. Vediamo alcuni messaggi comuni:

- I messaggi WM_ riguardano le finestre; per esempio, viene inviato WM_COMMAND quando si fa clic sulla voce di un menu (anche le corrispondenti scorciatoie da tastiera producono lo stesso effetto) oppure quando scatta un evento di controllo (più precisamente, quando il controllo rimanda una notifica alla sua finestra genitore). I parametri del messaggio identificano il menu o il controllo che hanno provocato il messaggio e, se si tratta di un controllo, anche il suo handle.
- I messaggi EM_ sono utilizzati dalle applicazioni per comunicare con i controlli di modifica; per esempio, un'applicazione invia un messaggio EM_GETLINE per copiare una riga di testo da un controllo e metterla in un buffer.
- I messaggi LB_ sono utilizzati dalle applicazioni per interagire con le caselle di riepilogo; per esempio, un'applicazione invia un messaggio LB_GETITEMRECT per sapere le dimensioni del rettangolo che circonda una voce visualizzata in una casella di riepilogo.

Naturalrnente esistono molti altri tipi di messaggi. L'applicazione Spy, già discussa in una precedente sezione di questo capitolo, consente di avere un'idea della varietà di messaggi che Windows e le sue applicazioni sono in grado di inviare e

ricevere. Anche gli esempi SubClass.Vbp e Tray.Vbp, che vedremo più avanti in questa sezione, consentono di conoscere il significato di diversi tipi di messaggio, e come utilizzarli.
Nei primi tempi delle GUI, prima che nascessero gli ambienti di sviluppo visuale di Windows, una parte imFortante di tutti i programmi Windows era costituita da una serie di gigantesche istruzioni condizionali che dirigevano correttamente ogni messaggio di Windows in arrivo. Questa codifica doveva naturalmente prevedere meccanismi di risposta ai messaggi (un processo noto con il nome di "smistamento dei messaggi"). La routine collegata a una finestra che si occupava dello smistamento dei messaggi costituiva la "procedura della finestra" o *windowprocedure.*
Per esempio, se il messaggio in arrivo diceva che il mouse aveva trascinato verso l'esterno i bordi di un forni che si poteva ridimensionare, si doveva invocare il codice che calcolava la nuova larghezza e altezza e disegnava il forni secondo le nuove dimensioni.

Si può accedere al flusso di messaggi che vengono inviati a una data finestra utilizzando la funzione API CallWindowProc. *Mediante una tecnica detta "di subclassing", illustrata più avanti in questo capitolo, una volta intercettati i messaggi inviati a unform o a un controllo, è possibile scrivere il codice che estende o modifica il comFortamento dell'oggetto.*

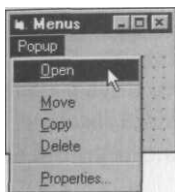
Come vedremo in due esempi successivi di questa sezione, CallWindowProc *riForta due argomenti imFortanti,* wParam *e l*Param. *In breve,* wParam *dice quale oggetto della finestra sta ricevendo il messaggio,* lParam *dice di che messaggio si tratta. Per esempio, wParam potrebbe dire che* Form1 *sta ricevendo un messaggio, che da* lParam *si può interpretare come un messaggio* WM_MOUSEMOVE, *relativo al movimento del mouse.*

Visual Basic consente di certo ai programmatori la creazione di applicazioni sofisticate senza il bisogno di conoscere nulla sui messaggi di Windows. Comunque, è sufficiente dedicare un po' di tempo all'apprendimento di questi messaggi per essere in grado di creare funzioni callback in VB6 che effettuano la sottoclassificazione del flusso di messaggi relativo a un oggetto, offrendo nuove potenti funzionalità che altrimenti non sarebbero disponibili in VB. Per vedere un esempio, si consideri l'applicazione Tray.Vbp più avanti in questa sezione.

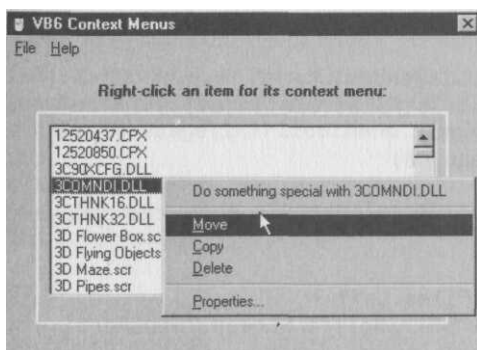
Aggiunta di menu di scelta rapida alle caselle di riepilogo

Questa dimostrazione riguarda il progetto presente nel CD-ROM allegato al libro con il nome Context. Vbp e illustra l'utilizzo dellAPISendMessage per aggiungere un menu di scelta rapida al contenuto di una casella di riepilogo (si veda la Figura 11.23). Oltre a SendMessage, *il progetto utilizza altre tre funzioni API, che non si sono ancora incontrate:* GetMenu, GetSubMenu *e* TrackPopUpMenu. *Vediamo le dichiarazioni di queste funzioni, tratte da API Dec.Bas:*

```
Declare Function SendMessage Lib "user32" Alias _
SendMessageA"    (ByVal hwnd As Long, _
  ByVal wMsg As Long, ByVal wParam As Long, _
  lParam As Any) As Long
Declare Function GetMenu Lib "user32" (ByVal hwnd As Long) As Long
Declare Function GetSubMenu Lib "user32" _
  (ByVal hMenu As Long, ByVal nPos As Long) As Long
Declare Function TrackPopupMenu Lib "user32" _
  (ByVal hMenu As Long, ByVal wFlags As Long, _
  ByVal x As Long, ByVal Y As Long, ByVal nReserved As Long, _
  ByVal hwnd As Long, Iprect As Any) As Long
```

Figura 11-23 *Questa applicazione illustra come aggiungere menu di scelta rapida a una casetta di riepilogo.*

- SendMessage invia un messaggio a una finestra, richiama la procedura relativa alla finestra di destinazione e non restituisce nulla fino a quando la procedura non ha elaborato il messaggio.
- GetMenu recupera l'handle del menu che appartiene alla finestra specificata.
- GetSubMenu recupera l'handle del menu a discesa che viene attivato dalla specifica voce del menu.
- TrackPopUpMenu visualizza un menu a discesa mobile in corrispondenza della posizione specificata e tiene traccia della selezione di voci nel menu a discesa. Questo menu può comparire in un punto qualsiasi dello schermo.

Per definire questo progetto si aggiunge un controllo FileListBox al menu di avvio (frmMain) e, utilizzando Menu Editor, si aggiunge una struttura di menu (PopUpMenu) a frmMenu (si veda la Figura 11.25). A questo punto si aggiungono al progetto le dichiarazioni e i moduli di codice API. In questo caso particolare, ho aggiunto lo stesso AboutBox illustrato in precedenza in questo capitolo.

Figura 11.24 *Si utilizza un modulo separato per creare la struttura del menu a discesa.*

Per quanto riguarda l'evento di caricamento di frmMain, ho impostato la proprietà .Path del controllo FileList alla directory System, utilizzando la tecnica illustrata in precedenza in questo capitolo per trovare la sua posizione. La definizione della proprietà .Path del controllo FileList fa sì che venga caricato con le voci nella directory specificata.

```
Private Sub Form_Load()
   Dim Buffer As String, x As Long
   Buffer = Space(255)
   x = GetSystemDirectory(Buffer, Len(Buffer))
   lstFiles.Path = ConvertCToVBString(Buffer)
   CenterForm Me
End Sub
```

Il menu di scelta rapida viene aperto da un clic del mouse, attivato dall'evento MouseDown. La procedura che richiama il menu a discesa è quindi sistemata nella procedura lstFiles_MouseDown. Per cominciare, viene ignorato tutto quello che non è un clic destro del mouse:

```
Private Sub lstFiles_MouseDown(Button As Integer, _
   Shift As Integer, x As Single, Y As Single)
   Dim hMenu As Long, hSubMenu As Long, ResultVal As Long, _
   MenuHeight As Long, MenuWidth As Long, MenuString As String, _
   IndexLst As Long, ItemRect As RECT
   If Button = vbRightButton Then
```

Si utilizzano quindi la coordinata dell'altezza passata dalla procedura MouseDown e una funzione di nome GetIndex (che vedremo più avanti) per selezionare la voce dell'elenco su cui si è fatto clic (dato che un semplice clic destro non esegue questa operazione) e lo si assegna come valore corrente alla proprietà .ListIndex dell'elenco.

```
IndexLst = GetIndex(Y)
'un clic destro su un elemento dell'elenco non lo seleziona
lstFiles.ListIndex = IndexLst
```

A questo punto si predispone una stringa di menu che dipende dal nome della voce su cui si è fatto clic destro. Si utilizzano le funzioni GetMenu e GetSubMenu per recuperare l'handle della prima voce di menu sotto PopUpMenu (si veda la Figura 11.23):

```
If lstFiles.List(lstFiles.ListIndex) <> "" Then
 MenuString = lstFiles.List(lstFiles.ListIndex)
 hMenu = GetMenu(frmMenu.hwnd)
 hSubMenu = GetSubMenu(hMenu, 0)
```

Si trova la fine del testo relativo alla casella di riepilogo (che deve essere sistemato nel menu di scelta rapida):

```
MenuWidth = (Me.Left + lstFiles.Left + _
            Me.TextWidth(MenuString)) \ Screen.TwipsPerPixelX _
            + 20
```

Si utilizza ora SendMessage (sì, siamo finalmente arrivati alla API SendMessage) per inviare un messaggio LB_GETITEMRECT a lstFiles. Non c'è ovviamente motivo per cui il codice dell'evento di un controllo non possa inviare messaggi al controllo stesso.

```
    ResultVal _SendMessage(lstFiles.hwnd, LB_GETITEMRECT, _
  IndexLst,   ItemRect)
```

S' utilizza LB_GETITEMRECT per determinare le dimensioni del rettangolo che circonda una voce dell'elenco; in effetti, sarebbe necessario conoscere solo la cima del rettangolo:

```
MenuHeight = (Me.Top + (Me.Height - Me.ScaleHeight) + _
 lstFiles.Top) \ Screen.TwipsPerPixelY + ItemRect.Top
```

A questo punto ho modificato il testo della voce di menu e, utilizzando i parametri altezza e *larghezza* già calcolati, si richiama l'API TrackPopUp per far scendere il menu.

```
          frmMenu!mnuPopOpen.Caption = _
             "Fa qualcosa di particolare con " & MenuString
          'fallo comparire!
          ResultVal = TrackPopupMenu(hSubMenu, TPM_LEFTALIGN, _
             MenuWidth, MenuHeight, 0&, frmMenu.hwnd, ByVal 0&)
          'aspetta l'azione dell'utente
          MsgBox "Metti il codice di ogni voce nel form di menu", _
             vbInformation, "Context"
End If
End If
End Sub
```

L'unica cosa che rimane da spiegare riguarda la funzione GetIndex. Vediamone la codifica:

```
Private Function GetIndex(Y As Single) As Long
   'La funzione da il ListIndex per la casella di riepilogo
   'appena l'utente preme il mouse alla coordinata y
  Dim OurRect As RECT, ResultVal As Long, Index As Long
   'Imposta il valore di ritorno di default
  GetIndex = -1
  If lstFiles.ListCount < 1 Then Exit Function
  Index = SendMessage(lstFiles.hwnd, LB_GETTOPINDEX, 0, 0&)
  ResultVal = SendMessage(lstFiles.hwnd, LB_GETITEMRECT, Index, _
     OurRect)
  If ResultVal <> 0 Then
     GetIndex = Index + (Y \ (Screen.TwipsPerPixelY * _
        OurRect.Bottom))
  End If
End Function
```

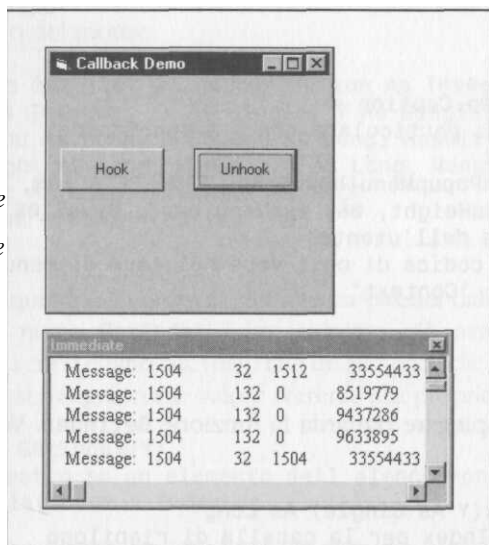
GetIndex riceve la coordinata dell'altezza relativa alla posizione corrente del mouse, così come la si ricava dalla procedura dell'evento MouseDown. Questa riporta il **.ListIndex** della voce nella casella di riepilogo. Se si esamina la codifica, si può

notare che si *utilizza,* per due volte la funzione SendMessage: una prima volta per rilevare .ListIndex della voce visibile in alto nell'elenco con un messaggio LB_GETTOPINDEX, una seconda volta per sapere *l'altezza* di una voce dell'elenco con il messaggio LB_GETITEMRECT.

Intercettazione del flusso di messaggi

È possibile intercettare i messaggi che vengono inviati a un forni mediante una sottoclassificazione della classe che si utilizza per creare il forni.

La Figura 11.25 mostra i messaggi di Windows intercettati durante l'invio a un form, *visualizzati nel pannello* Immediate *di VB. Si utilizzano i comandi* Debug. Print *per generare questa visualizzazione. Il progetto dimostrativo si trova nel CD-ROM allegato al libro con il nome di Subclass. Vbp.*

Figura 11.25 *È possibile intercettare il flusso di messaggi Windows che sono inviati a una particolare finestra mediante sottodassificazione (subdassing) della finestra.*

È ovvio che i messaggi numerici non sono molto significativi; per utilizzarli, bisogna conoscere il significato dei valori che esprimono. Per esempio, il valore corrispondente a un messaggio che dice che il mouse è stato spostato sopra una finestra o un controllo ha valore &H200. Si può definire una costante:

```
Public Const WM_MOUSEMOVE = &H200
```

e poi verificare l'uguaglianza della costante, adottando le azioni appropriate nel caso in cui si verifichi che il messaggio corrisponde alla costante. Vedremo come fare nel prossimo esempio, che aggiunge un 'icona nel vassoio di Windows. Le costanti relative ai messaggi e i loro valori si possono trovare mediante l'applicazione API Text Viewer che viene distribuita con Visual Studio 6.
Bisogna comprendere alcuni aspetti della questione prima di vedere come lavora l'esempio della sottoclassificazione. In primo luogo, una funzione *callback, o "cai-*

lback" indica una funzione definita dall'utente che gestisce una serie di valori generati da unafunzione API. Sipuò utilizzare laparola chiave AddressOf, *novità della versione 6 di VB, perfare riferimento all'indirizzo di unafunzione utilizzata come callback. La funzione callback deve avere lo stesso insieme di argomenti richiesto dall'originalefunzione API. Per esempio, l'istruzione*

```
lpPrevWndProc = SetWindowLong(gHW, GWL_WNDPROC, _
   AddressOf WindowProc)
```

dice a VB che lafunzione utente WindowProc vienepassata all'AP7 SetWindowLong *come suo argomento.* AddressOf *serve da puntatore alla funzione; ecco perché attesta corrispondenza prende a volte il nome di "puntatore difunzione".*

L'utilizzo delle callback e dellaparola chiave AddressOf *è soggetto aparecchie limitazioni. Ipuntatori di funzione devono essere tutti all'interno dello stesso modulo .Base nonpossono trovarsi in unform o in un modulo di classe. Non èpossibile utilizzare* **AddressOf** *per fare riferimento a una funzione esterna al progetto corrente.*

Si deve anche tenere conto che una volta che si comincia a fare riferimento a funzioni utente mediante un puntatore, per esempio utilizzando l'indirizzo di memoria, ci si lascia alle spalle la protezione dell'ambiente di sviluppo di Visual Basic. Il più piccolo errore di sintassi può provocare un blocco di VB. Fate quindi particolare attenzione quando lavorate con i puntatori di funzione; salvate spesso il vostro lavoro e fate delle copie di riserva.
Vediamo le dichiarazioni relative al modulo SubClass.Bas:

```
Declare Function CallWindowProc Lib "user32" Alias _
   "CallWindowProcA" (ByVal lpPrevWndFunc As Long, _
   ByVal hwnd As Long, ByVal Msg As Long, _
   ByVal wParam As Long, ByVal lParam As Long) As Long
Declare Function SetWindowLong Lib "user32" Alias _
   "SetWindowLongA" (ByVal hwnd As Long, _
   ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Public Const GWL_WNDPROC = (-4)

Global lpPrevWndProc As Long
Global gHW As Long
```

WindowProc è la funzione callback utilizzata per addentrarsi nel flusso di messaggi, che utilizza la parola chiave AddressOf:

```
Function WindowProc(ByVal hw As Long, ByVal uMsg As Long, _
   ByVal wParam As Long, ByVal lParam As Long) As Long
   Debug.Print "Message: "; hw, uMsg, wParam, lParam
   WindowProc = CallWindowProc(lpPrevWndProc, hw, _
      uMsg, wParam, lParam)
End Function

Public Sub Hook()
   lpPrevWndProc = SetWindowLong(gHW, GWL_WNDPROC, _
      AddressOf WindowProc)
End Sub
```

```
Public Sub Unhook()
   Dim tmp As Long
   tmp = SetWindowLong(gHW, GWL_WNDPROC, _
      lpPrevWndProc)
End Sub
```

Per fare una prova, inizializzate la variabile che contiene l'handle del form che verrà sottoclassificato in corrispondenza dell'evento di caricamento del forni:

```
Private Sub Form_Load()
   gHW = Me.hwnd
End Sub
```

Chiamate Hook per avviare la sottoclassificazione, Unhook per fermarla:

```
PrivateSubcmdHook_Click()
   Hook
End Sub

Private Sub cmdUnHook_Click()
   Unhook
End Sub
```

Inserimento di un'icona nel vassoio di Windows 95/98

La Figura 11.26 mostra la barra delle applicazioni di Windows. La zona all'estrema destra, dove compaiono le icone dell'orologio, dell'altoparlante e di 3Com, è nota come il "vassoio".

Figura 11.26

In questa sezione vedremo come sistemare l'icona di un'applicazione nel vassoio. Il codice dell'esempio si trova nel CD-ROM allegato al libro nel progetto Tray.Vbp. Vedremo come si risponde ai messaggi di Windows ricevuti dall'icona. In risposta a questi messaggi è possibile visualizzare un form quando l'utente fa doppio clic sull'icona nel vassoio, oppure visualizzare un menu di scelta rapida, e altro ancora.

Il semplicefatto che si possa sistemare l'icona della propria applicazione nel vassoio non significa che bisogna farlo per forza. Questo può risultare comodo per applicazioni quali una protezione residente contro i virus che sia sempre in esecuzione, oppure nel caso di utility particolari come il controllo del volume di una scheda audio.

Il primo passo consiste nel controllare, mediante le dichiarazioni e le tecniche illustrate in precedenza in questo capitolo, che sia in esecuzione Windows *95* oppure Windows NT 4 (o versioni successive), altrimenti niente vassoio! Il progetto comincia da Sub Main nel modulo Tray.Bas perché un'applicazione nel vassoio non deve

Hi solito visualizzare alcun form fino a quando l'utente non la attiva. Vediamo il codice di Sub Mairi:

```
Public Sub Main()
   Dim OS As OSVERSIONINFO
   OS.dwOSVersionInfoSize = Len(OS)
   GetVersionEx OS
   If QS.dwMajorVersion < 4 Then
       MsgBox "Mi dispiace, servono Windows 95/98 o NT4 o sucessivi!", _
        vbInformation, "Questo programma è andato!"
      End
   Else
      Debug.Print "OK. Ci siamo! 95 o NT4..."
   End If
   Load frmTray
End Sub
```

Il form dell'applicazione nel vassoio, frmTray, ha la proprietà Vislble impostata a False in fase di progettazione, in modo che non venga visualizzato nullo fino a quando non si è pronti.
Torniamo indietro per un istante; il Listato 11.6 mostra le dichiarazioni nel modulo Tray.Bas, diverse da quelle collegate alla determinazione della versione di sistema operativo:

Listato 11.16 *Dichiarazioni nel modulo Tray.Bas.*

```
Type NOTIFYICONDATA
   cbSize As Long
   hwnd As Long
   uID As Long
   uFlags As Long
   uCallbackMessage As Long
   hIcon As Long
   szTip As String * 64
End Type

Public Const WMJJSER = &H400
Public Const cbNotify& = WMJJSER + 42
Public Const uID& = 61860
Public myNID As NOTIFYICONDATA

Declare Function ShellNotifyIcon Lib "shell32.dll" _
   Alias "Shell_NotifyIconA" (ByVal dwMessage As Long, _
   lpData As NOTIFYICONDATA) As Long
    Public Const NIM_ADD=&H0
    Public Const NIM_DELETE=&H2
    Public Const NIM_MODIFY=&H1
    Public Const NIF_MESSAGE=&H1
    Public Const NIF_ICON=&H2
    Public Const NIF_TIP=&H4
        PublicConstWM_MOUSEMOVE=&H200
        PublicConstWM_LBUTTONDOWN=&H201
        PublicConstWM_LBUTTONUP=&H202
    PublicConstWMLBUTTONDBLCLK=&H203
```

```
Public Const WM_RBUTTONDOWN = &H204
Public Const WM_RBUTTONUP = &H205
Public Const WM_RBUTTONDBLCLK = &H206
Public Const WM_MBUTTONDOWN = &H207
Public Const WM_MBUTTONUP = &H208
Public Const WM_MBUTTONDBLCLK = &H209

Declare Function CallWindowProc Lib "user32" Alias _
   "CallWindowProcA" (ByVal lpPrevWndFunc As Long, _
   ByVal hwnd As Long, ByVal Msg As Long, _
   ByVal wParam As Long, ByVal lParam As Long) As Long
Declare Function SetWindowLong Lib "user32" Alias _
   "SetWindowLongA" (ByVal hwnd As Long,
   ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Public Const GWLJVNDPROC = (-4)

Global lpPrevWndProc As Long
Global gHW As Long
```

Il tipo NOTIFYICONDATA, l'API ShellNotifyIcon, e le costanti che iniziano con NIF_ sono utilizzati per manipolare il vassoio di Windows. Come abbiamo già visto in questa sezione, le costanti che iniziano con WM_ rappresentano specifici messaggi di Windows. La costante cbNotify& è definita dall'utente e viene utilizzata per identificare questa applicazione nella funzione callback. Ci si deve assicurare che sia superiore a WM_USER, il maggiore dei messaggi predefiniti di Windows; l'ho definita uguale a WM_USER + 42. In modo analogo ulD& costituisce l'identificatore nella struttura che sistema l'icona nel vassoio. Inoltre, il modulo contiene le procedure Hook e Unhook, già incontrate nell'esempio precedente relativo alla sottoclassificazione. In questo caso WindowProc, che vedremo fra un istante, è un po' differente dato che contiene il codice che risponde ai messaggi generati dall'utente.

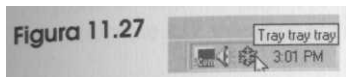
```
Public Sub Hook()
   lpPrevWndProc = SetWindowLong(gHW, GWL_WNDPROC, _
      AddressOf WindowProc)
End Sub

Public Sub Unhook()
   Dim tmp As Long
   tmp = SetWindowLong(gHW, GWL_WNDPROC, _
      lpPrevWndProc)
End Sub
```

Quando Sub Main carica il forni, viene eseguito il codice relativo al suo evento Load (mentre rimane invisibile):

```
Private Sub Form_Load()
   gHW = Me.hwnd
   myNID.cbSize = Len(myNID)
   myNID.hwnd = gHW
   myNID.uID = uID
   myNID.uFlags = NIF_MESSAGE Or NIF_TIP Or NIF_ICON
   myNID.uCallbackMessage = cbNotify
   myNID.hIcon = Me.Icon
```

```
    myNID.szTip = Me.Caption & Chr(0)
    ShellNotifyIcon NIM_ADD, myNID
    Me.Move (Screen.Width - Me.Width) \ 2, _
      '(Screen.Height - Me.Height) \ 2
    Hook
End Sub
```

Questo codice riempie l'istanza della struttura NOTIFYICONDATA, chiama l'API ShellNotifyIcon e sistema l'icona del form (in questo caso, un fiocco di neve) nel vassoio, come mostrato nella Figura 11.27. Il testo di suggerimento visibile in figura è definito dalla didascalia del form in questo codice, ma si può scrivere naturalmente quello che si vuole.

Figura 11.27

Prima di concludere, il codice di caricamento del form centra il form sullo schermo (anche se è ancora invisibile) e richiama la funzione callback dei messaggi di Windows. Il Listato 11.17 mostra il contenuto di WindowProc:

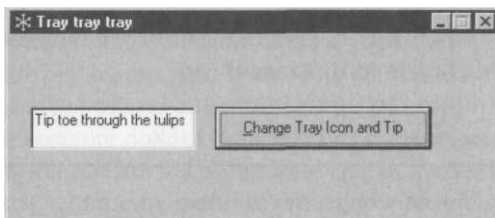
Listato 11.17 *La funzione WindowProc.*

```
Function WindowProc(ByVal hw As Long, ByVal uMsg As Long, _
    ByVal wParam As Long, ByVal lParam As Long) As Long
    If wParam = uID Then
        Select Case lParam
            Case WM_MOUSEMOVE
                Debug.Print "Spostamento del mouse"
            Case WMJ.BUTTONDOWN
                Debug.Print "Pressione pulsante sinistro"
            Case WM_LBUTTONUP
                Debug.Print "Rilascio pulsante sinistro"
            Case WM_LBUTTONDBLCLK
                Debug.Print "Doppio clic pulsante sinistro"
                'Visualizza il form
                frmTray.Visible = True
                AppActivate frmTray.Caption
            Case WM_RBUTTONDOWN
                Debug.Print "Pressione pulsante destro"
                'visualizza il menu popup
                frmTray.PopupMenu frmTray.mnuThing, _
                    vbPopupMenuRightAlign, , , frmTray.mnuHer
            Case WM_RBUTTONUP
                Debug.Print "Rilascio pulsante destro"
            Case WM_RBUTTONDBLCLK
                Debug.Print "Doppio clic pulsante destro"
                ChangeTray
            Case WM_MBUTTONDOWN
            Case WM_MBUTTONUP
            Case WM_MBUTTONDBLCLK
            Case Else
                Debug.Print "Messaggio sconosciuto: " & lParam
```

```
      End Select
   End If
   WindowProc = CallWindowProc(lpPrevWndProc, hw, _
      uMsg, wParam, lParam)
End Function
```

Vediamo come funziona. Per esempio, se l'utente fa doppio clic sull'icona nel vassoio con il pulsante sinistro del mouse, la funzione callback sa che è stato inviato un messaggio alla "nostra" applicazione dato che wParam=ulD. Se LParam = WM_-LBUTTONDBLCLCK, la procedura sa che è stato fatto un doppio clic sinistro sulla "nostra" icona. Il codice rende quindi visibile il forni e lo attiva (si veda la Figura 11.28):

```
frmTray.Visible = True
AppActivate frmTray.Caption
```

Figura 11.28 *L'utente rende visibile il modulo facendo doppio clic sull'icona nel vassoio.*

Gli altri eventi sono intercettati nello stesso modo e possono essere utilizzati per aggiungere codice che modifichi l'applicazione nel vassoio. Per esempio, si può cambiare l'icona nel vassoio e il suggerimento corrispondente nel caso in cui l'utente faccia doppio clic con il pulsante destro del mouse, come mostrato in Figura 11.29.

```
Case WM_RBUTTONDBLCLK
   ChangeTray

Public Sub ChangeTray()
   frmTray.Icon=frmTray.Label1.DragIcon
   myNID.hIcon = frmTray.Icon
   myNID.szTip = Trim(frmTray.Textl.Text)
   ShellNotifyIcon NIM_MODIFY, myNID
End Sub
```

La procedura ChangeTray viene richiamata sia quando l'utente fa doppio clic sull'icona nel vassoio sia dal pulsante di comando presente nel forni stesso del vassoio.

La nuova icona (il sole al posto del fiocco di neve) è memorizzata nella proprietà DragIcon *di un controllo d'etichetta invisibile, che costituisce un sistema a basso consumo di risorse che consente di inserire icone aggiuntive nei propri form.*

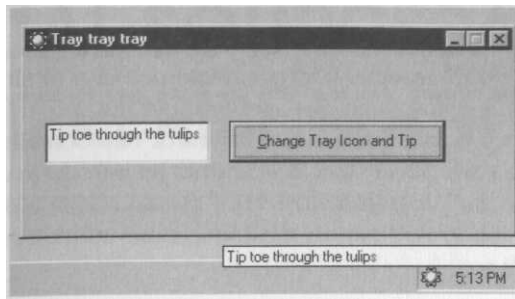
Figura 11.29 *Si utilizza l'Api ShellNotifyIcon per modificare nel vassoio un 'icona e il suggerimento corrispondente.*

È pratica comune aggiungere un menu di scelta rapida alle applicazioni posizionate come icone nel vassoio, come mostrato in Figura 11.30.

Figura 11.30 *Menu di scelta rapida per un'applicazione nel vassoio.*

È stato aggiunto al forni il menu di nome mnuThing; questo è richiamato da:

```
Case WM_RBUTTONDOWN
frmTray.PopupMenu frmTray.mnuThing, _
   vbPopupMenuRightAlign, , , frmTray.mnuHer
```

I menu di questo tipo sono illustrati nel Capitolo 18.
È estremamente imFortante rilasciare la funzione callback e cancellare l'icona nel vassoio quando si scarica l'applicazione:

```
Private Sub Form_Unload(Cancel As Integer)
   Unhook
   ShellNotifyIcon NIM_DELETE, myNID
End Sub
```

Un'ultima questione: probabilmente si desidera che gli utenti possano minimizzare l'applicazione nel vassoio. Un'applicazione così minimizzata visualizza solo la sua icona nel vassoio, senza presentare anche una finestra minimizzata. Per ottenere questo comFortamento, quando l'utente minimizza il modulo si deve ripristinare la sua dimensione normale e renderlo invisibile. Il codice seguente, che si trova nell'evento di ridimensionamento del form, si occupa si questo:

```
Private Sub Form_Resize()
   If Me.WindowState = vbMinimized Then
      Me.Hide
      Me.WindowState = vbNormal
   End if
End Sub
```

Riepilogo

Questo capitolo si è occupato di argomenti strettamente pratici e di altre faccende puramente teoriche. Abbiamo visto quello che serve sapere per iniziare a utilizzare gli strumenti di Visual Studio 6.0, tra i quali ActiveX Control Test Container, API Text Viewer, Spy++ e WinDiff. La parte successiva del capitolo ha introdotto alcune delle API utilizzate comunemente dai programmatori VB. Passando dalla teoria alla pratica, i progetti di esempio con le API hanno mostrato come utilizzare queste funzioni per una molteplicità di scopi.

- Avete visto come sistemare un modulo in primo piano.
- Avete visto come si spostano i controlli tra i moduli.
- Avete scoperto come bloccare gli utenti su un modulo.
- È stata trattata la modifica del menu *Window* di un'applicazione.
- Ho spiegato come si controllano le risorse di sistema.
- Avete visto come si crea un generico forni *About.*
- Vi ho mostrato come rilevare le directory Windows e System.
- Avete visto come controllare le finestre utilizzate nel thread corrente.
- Avete fatto pratica sull'utilizzo delle funzioni callback.
- Avete visto l'utilizzo della funzione API SendMessage per creare un menu di scelta rapida all'interno del controllo di una casella elenco.
- Avete visto come intercettare il flusso di messaggi di Windows.
- Avete visto come si sistema un'icona di programma nel vassoio di Windows e come si risponde ai messaggi generati dall'interazione dell'utente con questa icona.

VISUAL SOURCESAFE (ENTERPRISE EDITION)

- Visual SourceSafe Administrator.
- Utilizzo di Visual SourceSafe Explorer.
- Utilizzo di VSS Explorer per creare un progetto VSS locale.
- Integrazione di VSS con Visual Basic.
- Controllo dei file in uscita e in entrata.
- Operazione "diffing".

La versione Enterprise Edition di Visual Basic 6 prevede un'applicazione chiamata Microsoft Visual SourceSafe (VSS) la quale contiene a sua volta due programmi, VSS Administrator, per amministratori; e VSS Explorer, per programmatori.

Quando si installa VSS, conviene utilizzare la procedura Custom e verificare che sia attiva l'opzione Enable SourceSafe Integration. *In questo modo si collega automaticamente VSS a Visual Basic IDE, grazie a una nuova voce VSS che si aggiunge a quelle presenti nel menu* Tools *di VB6. È comunque sempre possibile installare SourceSafe in un momento successivo all'installazione iniziale di VB6.*

Visual SourceSafe costituisce un sistema di controllo della versione che è stato creato principalmente per lo sviluppo di software da parte di gruppi di programmatori. Se si lavora all'interno di un gruppo, è essenziale disForre di un sistema di controllo della versione in corso, per evitare che i membri del gruppo possano pestarsi i piedi.

Nello stesso tempo, VSS può risultare utile anche per il singolo programmatore. VSS è sostanzialmente una "blblioteca di prestiti" virtuale che dispone di un blbliotecario, VSS Administrator, e di file che possono essere "presi in prestito" dai programmatori che abbiano diritto di accesso al progetto con VSS Explorer. Questo capitolo introduce entrambi gli aspetti di Visual SourceSafe: l'amministrazione di VSS e il suo utilizzo dal punto di vista di un programmatore.

Visual SourceSafe Administrator

Si utilizza Visual SourceSafe Administrator per aggiungere utenti (programmatori) e per definire i corrispondenti privilegi di accesso ai diversi progetti VSS. Non è necessario *utilizzare* una password per avviare VSS Administrator, anche se un avviso suggerisce di crearne una per l'amministratore. Una volta inserita questa password, non bisogna dimenticarla altrimenti si è perduti; come afferma il manuale di Visual SourceSafe, "se si dimentica la password assegnata all'amministratore, non c'è alcun modo di eseguire Visual SourceSafe Administrator per modificarla … se si perde la password dell'amministratore, contattare Microsoft Technical Support Services per un'assistenza diretta". In caso di necessità, il numero diretto dell'assistenza è 027 70398398; tenete a Fortata di mano il numero di serie del vostro prodotto.

Per cominciare

SourceSafe è un'applicazione client/server. Il server corrisponde ad Administrator, il client a Explorer. Prima di utilizzare SourceSafe occorre installare il server. Si ricordi che l'installazione del server VSS non fa parte dell'installazione di Visual Studio 6, nemmeno come opzione disponibile. Visual Studio installa in modo predefinito il client VSS Explorer, ma non il server.
Se si lavora in un ambiente nel quale il server Visual SourceSafe è già installato e tutto è definito a puntino, non ci si deve preoccupare più di tanto. Se invece si deve installare il server, occorre eseguire il corrispondente programma di installazione.

Per installare il server di Visual SourceSafe, eseguire il programma di installazione che si trova nel secondo CD-ROM di Visual Studio 98, nella cartella Vssjss.

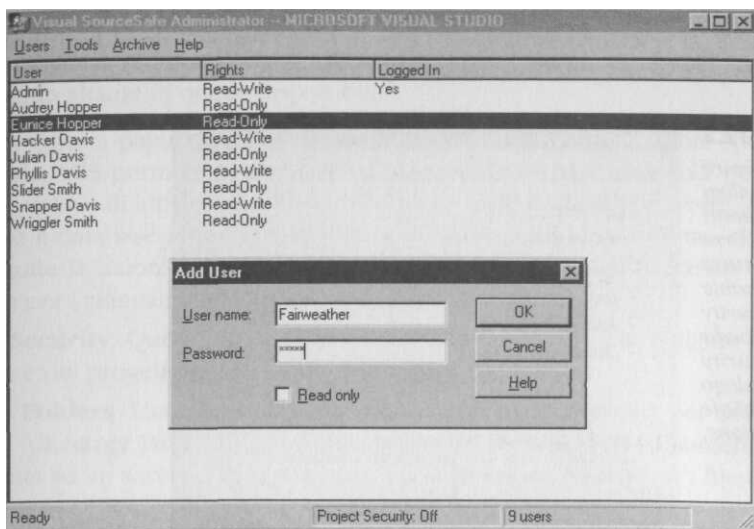
Per definizione, il server di Visual SourceSafe è impostato come database SourceSafe che memorizza il codice del progetto nella directory Microsoft Visual Studio\VSS. Se si vogliono inserire database di Visual SourceSafe aggiuntivi, si deve eseguire l'utility da riga di comando Mkss. Questo programma si trova nella directory del server di Visual SourceSafe, nel secondo CD-ROM di Visual Studio 98, nella directory Vss_ss\Vss\Win32.

Prima di eseguire Mkss per creare un database Visual SourceSafe, occorre creare una directory vuota per questo database. Per esempio, si utilizza Gestione risorse per creare una cartella C:\VssData; si esegue quindi Mkss a partire da una finestra DOS con il comando:

```
Mkss C: WssData
```

Avvio di Administrator

Una volta installato correttamente il server di Visual SourceSafe, è possibile far partire VSS Administrator dal menu del programma. Lanciato questo, l'applicazione Visual SourceSafe Administrator risulta simile alla finestra mostrata in Figura 12.1, anche se non sono presenti altri utenti ad eccezione dell'amministratore e di un ospite.

Figura 12.1 *La finestra dell'applicazione Visual SourceSafe Administrator mostra otto utenti (oltre l'amministratore). A questo punto è possibile inserire un nuovo utente.*

Inserimento di utenti

Si seleziona la prima voce della barra di menu mostrata in Figura 12.1, *Users,* per aggiungere utenti, per cancellarli oppure per modificarne le password. Per aggiungere un nuovo utente, selezionare *Add User* dal menu *Users;* compare il dialogo *Add User* mostrato in Figura 12.2. A questo punto si possono inserire i nomi degli utenti, le password e, se lo si desidera, limitare l'accesso degli utenti in sola lettura. Se si seleziona *Read Only,* l'utente dispone di accesso in sola lettura a qualunque progetto VSS. Questa opzione definisce un accesso valido in sola lettura di tipo permanente. Se si vuole invece limitare i privilegi di accesso che riguardano progetti specifici e garantire invece l'accesso completo ad altri progetti, si devono definire questi privilegi utilizzando il dialogo *ProjectRights* discusso nella prossima sezione.

Modifica dei privilegi di accesso a un progetto

La seguente procedura consente di definire i privilegi di accesso relativi a ciascun utente:

1. Si deve creare un progetto Visual SourceSafe prima di definire i privilegi di accesso relativi a quel progetto. I progetti VSS (che non si devono confondere con i progetti Visual Basic, si tratta di faccende diverse tra loro) vengono creati utilizzando Visual SourceSafe Explorer oppure nello stesso Visual Basic. Si noti che in entrambi i casi non ci si può collegare a VSS Explorer se non si è abilitati a farlo come utenti tramite Administrator. La creazione di un progetto VSS con entrambi i metodi è trattata più avanti in questo capitolo.

2. A questo punto si apre il dialogo *SourceSafe Options* mostrato in Figura 12.2 (si seleziona *Options* dal menu *Tools)* e si seleziona l'opzione *Enable Project Security* nella scheda *Project Security*.

Figura 12.2

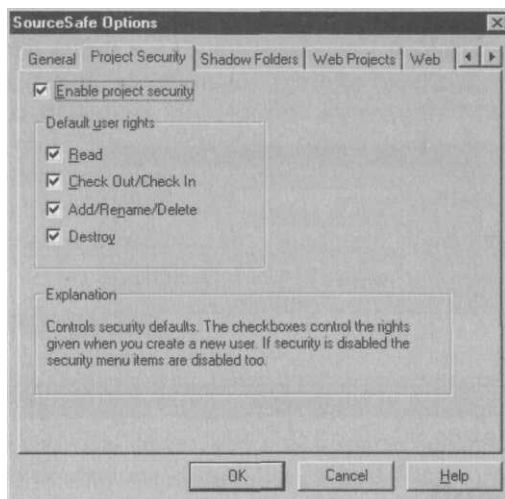
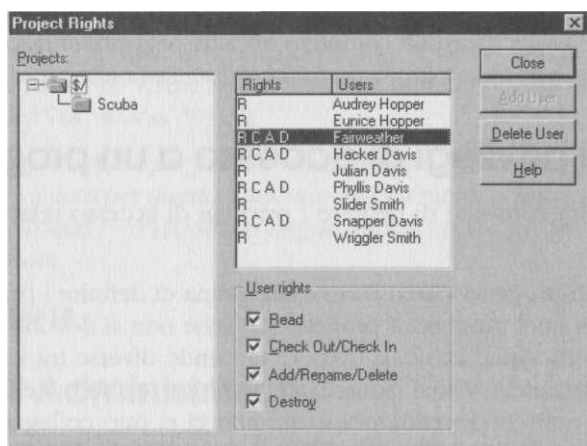
Per definire i privilegi degli utenti occorre selezionare l'opzione Enable Project Security nella scheda Project Security del dialogo SourceSafe Options.

Da qui, selezionare *Rights* dalla voce *Project* del menu *Tools;* compare il dialogo *Project Rights* mostrato in Figura 12.3. Come si può vedere dalla figura, alcuni utenti dispongono di accesso completo mentre altri si trovano in sola lettura per quanto riguarda il progetto selezionato Scuba. Per aggiungere o togliere determinati privilegi, selezionare il nome dell'utente e modificarne i privilegi utilizzando le caselle di opzione visibili nella parte bassa della finestra di dialogo.

Figura 12.3

L'amministratore del progetto utilizza il dialogo Project Rights per definire i privilegi degli utenti relativi a progetti specifici. In questo caso, alcuni utenti possono solo accedere in lettura mentre altri dispongono di accesso completo.

Opzioni di Administrator

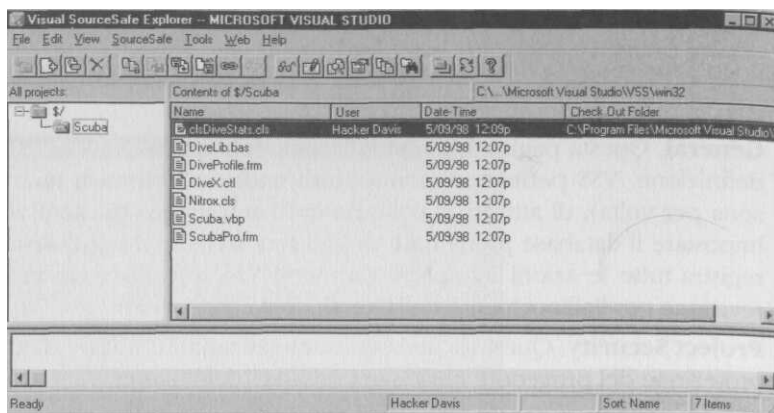
Il dialogo *SourceSafe Options* consente di definire altre opzioni relative all'amministratore; vediamo alcune di quelle disponibili:

- **General.** Questa pagina consente di abilitare controlli multipli di file (per definizione, VSS permette il controllo di un file da parte di una sola persona per volta), di attivare il collegamento automatico di utenti in rete, di impostare il database predefinito di VSS e di stabilire il log di sistema (che registra tutte le azioni intraprese da utenti VSS e fornisce un processo di revisione per l'allineamento della versione).
- **Project Security.** Questa scheda consente all'amministratore di abilitare la protezione del progetto e definisce i privilegi degli utenti.
- **Shadow Folders.** L'amministratore può utilizzare questa scheda per definire una "directory fantasma", una directory di accentramento solitamente posizionata su un server, che contiene le versioni più recenti di tutti i file di un progetto.
- **Web Projects.** L'amministratore può utilizzare questa scheda per definire le opzioni che riguardano un singolo progetto Web, opzioni che comprendono l'assegnamento di un progetto VB come sito Web, la definizione di un URL, la specifica relativa a una radice virtuale e quella di un percorso di ordinamento.
- **Web**. Si utilizza questa scheda per definire le opzioni che riguardano i progetti Web su cui si sta lavorando. Si può quindi specificare un proxy per l'installazione remota attraverso un firewall e definire il nome dei file predefinito per le pagine Web.
- **File Types.** Si utilizza questa scheda per definire i tipi predefiniti di file relativi agli elenchi di file che compaiono nei dialoghi utente di VSS. Per esempio, è possibile creare un gruppo di file per Visual Basic che includa i seguenti tipi di file: *.Bas, *.Cls, *.Frm, *.Frx, *.Vbp e *.Res.

In definitiva, l'applicazione Visual SourceSafe Administrator consente di definire gli utenti e le loro password, di definire i privilegi degli utenti e di creare una directory di progetto centralizzata. VSS Administrator funziona anche come rete di sicurezza per il progetto, nel senso che memorizza ogni versione di file in un database VSS dedicato. Nella prossima sezione vedremo come si utilizza Visual SourceSafe Explorer per gestire il controllo della versione di un progetto.

Utilizzo di Visual SourceSafe Explorer

Visual SourceSafe Explorer *è* stato creato tenendo presente le esigenze di un gruppo di programmatori al lavoro, ma può benissimo essere utile anche nel caso di chi opera da solo, in quanto tiene comunque traccia della versione che si evolve. Quando si lancia Visual SourceSafe Explorer, compare la finestra VSS Explorer, come mostrato in Figura 12.4.

Figura 12.4
Vediamo Visual SourceSafe Explorer con il progetto Scuba selezionato. VSS consente di coordinare tra loro i file del progetto.

VSS tiene traccia dei file del progetto e delle loro eventuali modifiche; se si utilizza Visual SourceSafe, è possibile tenere sotto controllo la storia di un file, riFortarsi su una versione precedente di un file e confrontare le differenze tra due versioni dello stesso file.

VSS genera una cartella virtuale di nome $/ come directory dipartenza dei progetti VSS da memorizzare.

Creazione di un progetto VSS mediante VSS Explorer

Il primo passo per definire il controllo della versione consiste nella creazione di un progetto Visual SourceSafe. È già stato ricordato che un progetto VSS non è la stessa cosa di un progetto VB; si tratta di due faccende differenti, anche se collegate tra loro.

Il progetto VSS è una raccolta di file di qualunque tipo, anche di quelli che Visual Basic non è in grado di riconoscere. Di conseguenza un progetto VSS può contenere un file che non fa parte di un progetto VB (per esempio, un modello di Word, un file C++ oppure un file HTML). Inoltre, un progetto Visual Basic può fare parte di un sottoprogetto Visual SourceSafe; un progetto VSS può contenere diversi progetti Visual Basic.

Per creare un nuovo progetto Visual SourceSafe, si evidenzia la cartella nel pannello sinistro di VSS Explorer nella quale si vuole sistemare il progetto. Si seleziona quindi *Create Project* dal menu *File,* in cima alla finestra di VSS Explorer. Nel dialogo che compare, mostrato in Figura 12.5, si scrive il nome del progetto e un commento qualsiasi che riguarda il progetto (i motivi della sua creazione, una descrizione e così via).

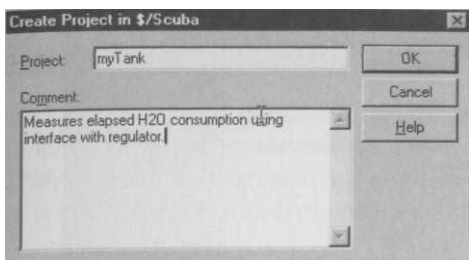
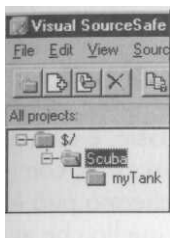
Figura 12.5
In Visual SourceSafe Explorer si possono creare nuovi progetti.

Una volta terminato, si fa clic su *OK;* il nuovo progetto compare nella finestra VSS Explorer (si veda la Figura 12.6).

Figura 12.6
Una volta creato un progetto, questo compare in ordine gerarchico nel pannello di sinistra di VSS Explorer.

A questo punto è possibile aggiungere file al progetto utilizzando la voce *Add Files* nel menu *File.* È anche possibile aprire Gestione risorse e trascinare i file all'interno del progetto. Se non sono ancora stati creati i file Visual Basic che devono entrare nel progetto, li si può aggiungere in un secondo tempo da Visual Basic, dato che Visual SourceSafe è integrato completamente con Visual Basic.

Integrazione di VSS con Visual Basic

Se siete destinati a diventare utenti di Visual SourceSafe (in altre parole, avete Visual SourceSafe Explorer installato sul vostro computer e l'amministratore del sistema vi ha inseriti come utenti VSS mediante Visual SourceSafe Administrator), sicuramente avete a disposizione Visual SourceSafe Add-In in Visual Basic.
Se, per un motivo qualsiasi, questo non è ancora stato caricato, è possibile predisporre VSS Add-In e definirlo in modo che parta automaticamente al lancio di VB, attivando l'opzione *Source Code Control Add-In* in *Add-In Manager,* come mostrato nella Figura 12.7.

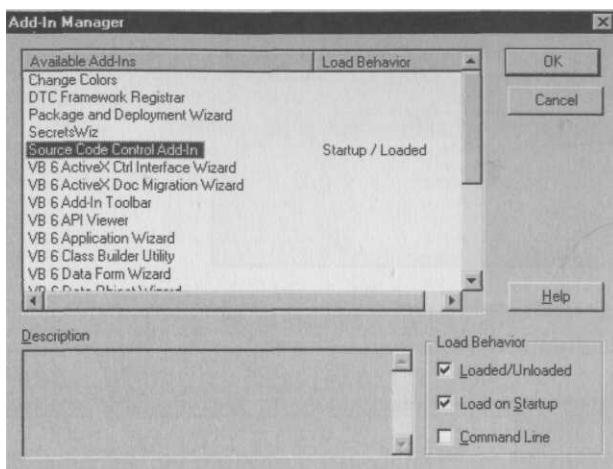
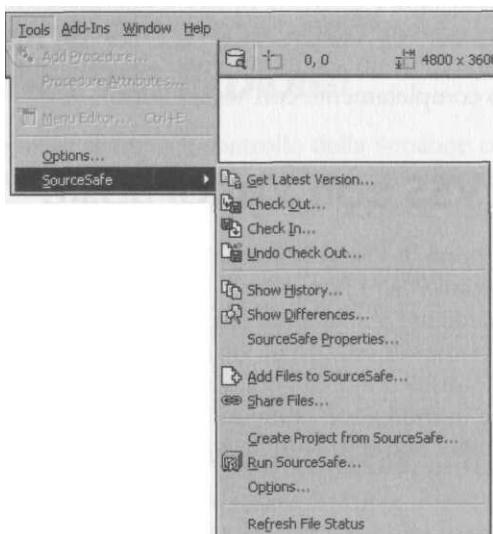
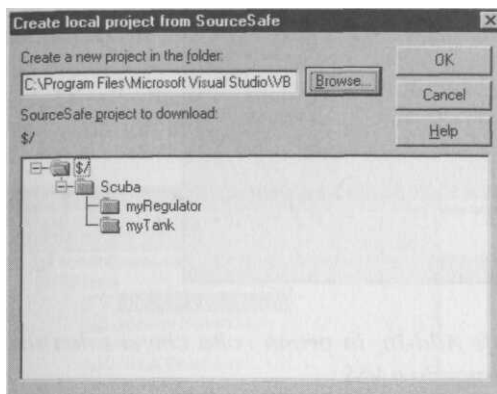
Figura 12.7
Se si carica Source Code Control Add-In in Add-In Manager, VSS risulta integrato con Visual Basic.

Per vedere se VSS è correttamente agganciato a Visual Basic, è sufficiente lanciare VB e controllare il menu *Tools.* Se compare una voce *SourceSafe,* come nella Figura 12.8, siete a posto. Il sottomenu che vedrete sul vostro schermo può essere diverso da quello mostrato nella Figura 12.8, perché dipende da quello che avete collegato a un progetto VSS.

Figura 12.8
Visual SourceSafe inserisce le proprie voci tra quelle disponibili nel menu Tools di Visual Basic.

Creazione di un progetto locale VSS con Visual Basic

È possibile anche creare un progetto VSS mentre si lavora in Visual Basic. È sufficiente selezionare *Create Project* dal sottomenu *SourceSafe* che si trova nel menu Tools di VB. Nel dialogo che compare, mostrato in Figura 12.9, selezionare il progetto VSS che si vuole creare localmente e fare clic su *OK*.

Figura 12.9 *È possibile creare un progetto VSS locale direttamente da Visual Basic.*

Inserimento di un progetto Visual Basic in VSS

Per aggiungere un progetto VB a Visual SourceSafe, si deve prima salvare il progetto in Visual Basic. Una volta che il progetto è stato salvato, se non è già aperto un progetto Visual SourceSafe è sufficiente selezionare *Add to SourceSafe* dal menu *SourceSafe*. Si utilizza il dialogo mostrato in Figura 12.10 per selezionare una posizione VSS per il progetto. Se risulta già aperto un progetto VSS, questo dialogo compare automaticamente quando si salva un progetto Visual Basic.

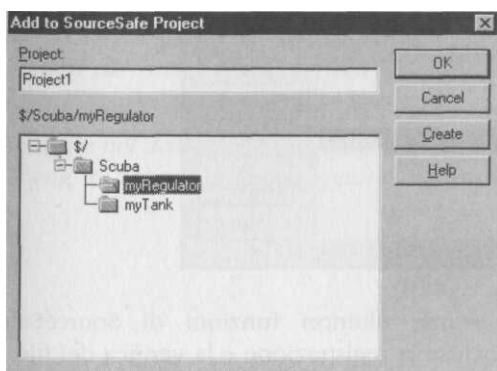
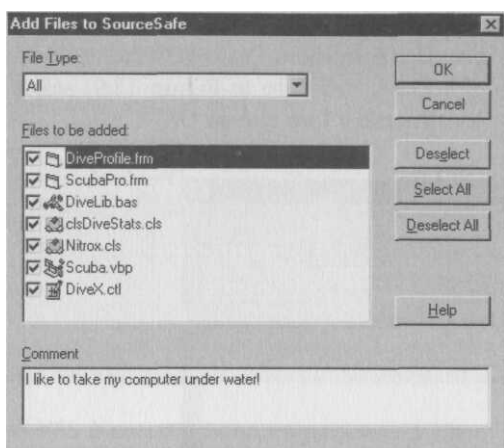
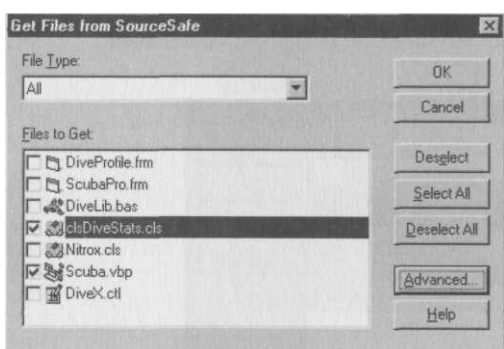
Figura 12.10 *Si utilizza il dialogo Add to SourceSafe Project per selezionare la posizione VSS relativa a un progetto di Visual Basic.*

Una volta selezionata una posizione per il progetto, si può decidere quali file aggiungere, come mostrato in Figura *12.11.*

Figura 12.11
Si può decidere quali file Visual Basic aggiungere a SourceSafe.

Se è stato caricato SourceSafe Add-In, la prima volta che si salva un progetto VB viene chiesto se si vuole aggiungerlo a VSS.

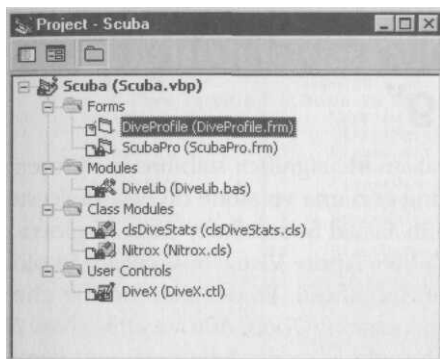
Determinazione della versione più recente di un file

Per reperire la versione più recente di un file in ambiente Visual Basic, selezionare *Get Latest Version* dal menu *SourceSafe.* Il dialogo *Get Files from SourceSafe,* mostrato in Figura 12.12, consente di selezionare i file desiderati.

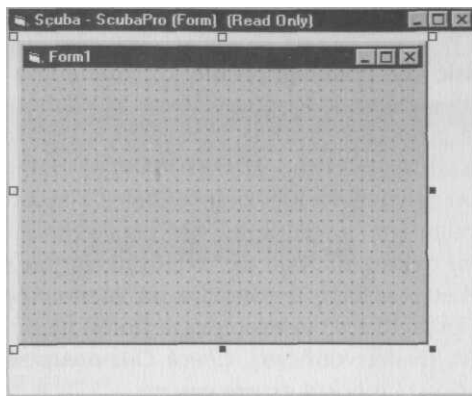
Figura 12.12
Si utilizza Get Files nel dialogo SourceSafe per ricercare la versione più recente dei file in Visual Basic.

È possibile accedere ad alcune ulteriori funzioni di SourceSafe a partire dall'ambiente Visual Basic, inclusa la registrazione e la verifica dei file trattate nella prossima sezione.

Registrazione e verifica dei file

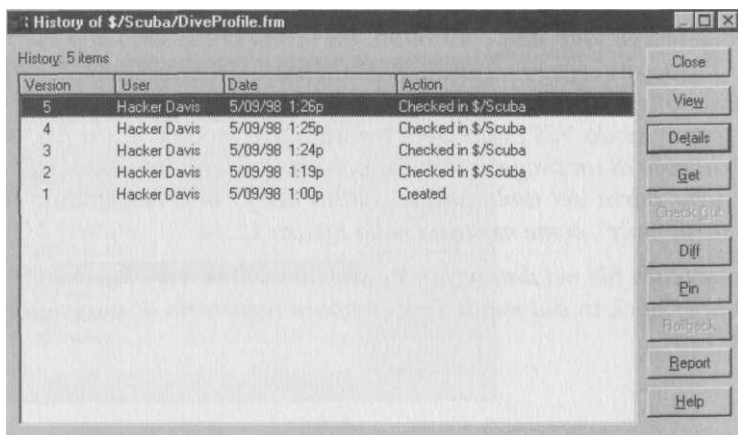
Nel caso di progetti Visual Basic che contengono file appartenenti a un progetto Visual SourceSafe, risultano differenziate le icone utilizzate in VB Project Explorer. Le icone relative a file collegati a un progetto VSS (per esempio, un forni) presentano in aggiunta un piccolo lucchetto *azzurro* visibile nell'angolo inferiore sinistro, Inoltre per chi è un utente con privilegi di lettura e scrittura, a sinistra della consueta icona di un modulo VB compare l'icona di una piccola pagina. Una volta che si è utilizzata la voce *Check Out* del menu *Tools* per verificare un file del database VSS scompare il piccolo lucchetto *azzurro* e compare un segno rosso di spunta nell'icona della pagina. Nella Figura 12.13, per esempio, il primo file in elenco nella finestra *Project,* DiveProfile.frm, risulta verificato. *Check Out* recupera il file nella forma a lettura e scrittura, in modo che possa essere variato.

Figura 12.13
Per verificare un file, selezionare Check Out nel menu SourceSafe. Nel Project Explorer visibile in figura è stato verificato il modulo DiveProfile.

La funzionalità principale di un sistema di controllo della versione, quale è Visual SourceSafe, consiste nella registrazione e nella verifica dei file. Si possono modificare solo i file che sono stati controllati. Una volta che si sono apFortati dei cambiamenti, i file devono essere registrati di nuovo. In un ambiente di sviluppo condiviso la registrazione e la verifica dei file prendono il posto delle operazioni di salvataggio. In questo modo VSS può tenere traccia delle modifiche sui file, e di chi le ha fatte. Ogni voce di un progetto VSS che non risulta verificata viene riferita come tale in VB; nella barra del titolo del progettista del form o del modulo compaiono le parole "Read Only", come mostrato nella Figura 12.14.

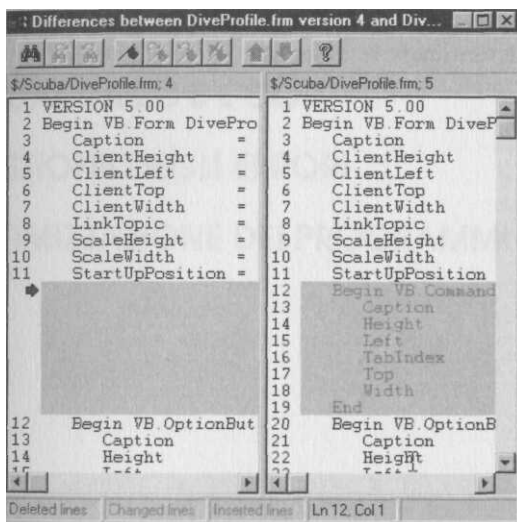
Per riFortare un file nel database VSS, selezionarlo in VB Project Explorer e quindi selezionare Check In *dal menu* Tools, *oppure registrarlo di nuovo utilizzando VSS Explorer.*

Figura 12.14
In Visual Basic un modulo che non è ancora stato verificato risulta marcato in sola lettura.

Individuazione delle modifiche su un file: operazione "diffing"

Eseguire un'operazione "diffing" su un file significa stabilire le differenze tra questo e un altro file. Di solito si fa il diffing con una versione diversa dello stesso file. Una volta che i file sono stati verificati in Visual SourceSafe, entra in gioco la potenza di controllo della versione, propria di VSS. Aprite Visual SourceSafe Explorer e selezionate il progetto i cui file sono stati modificati. Evidenziate un file che è stato registrato di nuovo e selezionate in successione *Tools, SourceSafe, Show History;* viene visualizzata una finestra che rappresenta la storia della versione, come mostrato in Figura 12.15.

Figura 12.15
La finestra History mostra le diverse versioni di un file che è stato più volte registrato e verificato.

La Figura 12.15 mostra la storia del file VB DiveProfile.frm. Come si può notare nella cronologia illustrata nella figura, il file è stato creato in data 05/09/98 alle ore 1:00 porneridiane, e siamo alla versione 1. La seconda voce indica che è stato registrato nuovamente in data 05/09/98 alle ore 1:19, e siamo alla versione 2. La terza voce,che riguarda la versione 3, mostra la registrazione effettuata nello stesso giorno alle ore 1:24, e così via.

Per vedere le differenze tra la seconda e la terza versione di questo file, si selezionano le due versioni e poi si fa clic sul pulsante *Diff.* Si apre la finestra mostrata in Figura 12.16, la quale evidenzia le differenze tra le due versioni all'interno di una casella grigia. A questo punto, si possono accettare le modifiche in un file lasciandolo così come è, oppure si può utilizzare il pulsante *RollBack* nella finestra *History* per ritornare a una versione precedente dello stesso file.

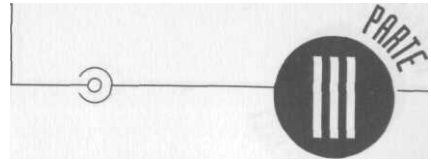
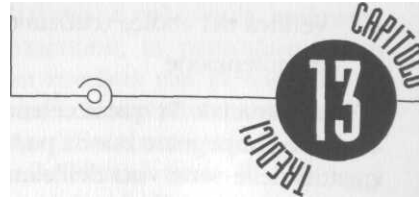
Figura 12.16
la finestra Diff mostra le differenze che esistono tra due versioni di un file.

Si utilizza il pulsante Pin *nella finestra History per "congelare" la versione di un file. Fino a quando non viene utilizzato Unpinned, nessuna modifica può essere apportata a questa versione.*

Riepilogo

Microsoft Visual SourceSafe è un sistema di controllo della versione che è stato ideato per proteggere i progetti complessi; questo sistema prevede la creazione di un database che contiene tutte le modifiche apportate a un file. È stato pensato tenendo conto delle esigenze dei gruppi di lavoro. VSS è costituito da due applicazioni: Visual SourceSafe Administrator, che definisce gli utenti ed i corrispondenti privilegi di accesso, e Visual SourceSafe Explorer, che tiene traccia delle differenti versioni dei file di un progetto che hanno luogo durante la sua definizione.

- In questo capitolo ho trattato l'installazione di un server Visual SourceSafe.

- Avete visto come si utilizza Visual SourceSafe Administrator per aggiungere utenti e definire le loro password.
- Ho spiegato come si utilizza Visual SourceSafe Administrator per modificare i privilegi di accesso al progetto.
- Avete visto come si utilizza Visual SourceSafe Administrator per definire le opzioni relative all'amministratore.
- Avete visto come si utilizza VSS Explorer per essere sicuri che VSS sia integrato perfettamente con Visual Basic.
- Ho spiegato come si *utilizza* VSS Explorer per creare un progetto Visual SourceSafe e come vi si aggiungono file.
- Avete visto l'utilizzo di VSS Explorer per la registrazione e la verifica dei file e per riFortare i file nella libreria VSS.
- Avete visto come si utilizza VSS Explorer per analizzare la storia delle versioni di un file e per determinare le differenze fra le versioni.

SEGRETI
DI PROGRAMMAZIONE

UNA BUONA PRATICA DI PROGRAMMAZIONE

- Mettere in pratica le migliori tecniche di programmazione
- Progettazione architettonica delle applicazioni di VB
- Convenzioni per l'attribuzione dei nomi
- Aggiunta di proprietà e metodi personalizzati ai form
- Creazione di uno stack in VB
- Interruzione dei cicli Do
- Gestione delle caselle di riepilogo
- Analisi e manipolazione delle stringhe (incluso il codice di VB)
- Arrotondamento automatico dei numeri
- Creazione di elenchi dei tipi di carattere

Questo capitolo inizia con un'analisi delle caratteristiche che contraddistinguono la buona programmazione: *chiarezza,* eleganza, solidità della progettazione e buon senso. Come è stato detto precedentemente, ritengo che sia necessario fornire degli esempi, e non solo descrivere con le parole; per questo motivo continuerò a presentare tecniche e trucchi di programmazione specifici per risolvere in modo semplice e adeguato i problemi comuni che sorgono in VB. In questo capitolo sono presentati gli strumenti che permettono di gestire in modo efficace molti problemi correlati alla programmazione in VB che si presentano ciclicamente.

La buona pratica di programmazione

Lo sviluppo di software è un processo estremamente complesso, per gestire il quale è necessario fare attenzione sia ai dettagli che alla situazione generale e che può essere suddiviso nelle seguenti attività separate:

- definizione di un problema
- progettazione dell'architettura
- progettazione dettagliata
- creazione del codice
- debugging
- verifica del codice (collaudo)
- manutenzione

Il punto cruciale in questo elenco è che la pianificazione, di qualsiasi tipo si tratti dovrebbe impegnare buona parte del tempo dello sviluppatore di software (almeno quattro delle sette voci dell'elenco precedente coinvolgono la pianificazione anziché la creazione effettiva del codice).
In realtà le società e i programmatori di norma si immergono in un progetto senza aver capito completamente cosa comForta. A causa di questa fretta eccessiva vengono tralasciate possibili soluzioni creative, magari migliori. Troppo spesso si tenta e poi si abbandona un approccio dopo l'altro, lasciandosi indietro enormi blocchi di codice che poi vengono chiamati "prototipi". Nei casi peggiori, improvvisamente si riceve una telefonata: "Dobbiamo avere quell'applicazione entro la fine della settimana" (o del mese, o dell'anno). A questo punto non si ha il tempo di pianificarla da capo e il "prototipo" più recente diventa la base della versione finale. Non ci si dovrebbe sorprendere se un progetto sviluppato in questo modo contiene un'infinità di bachi e non è facile da mantenere.
Il messaggio che vorrei trasmettere è che, indipendentemente dalle dimensioni del progetto, sia esso una piccola routine o un'applicazione composta da centinaia di form e di moduli, si dovrebbe allocare fin dall'inizio tanto tempo per la pianificazione quanto per la creazione del codice. È consigliabile pianificare al livello dove si può formulare la logica di ogni routine in pseudocodice addirittura prima di iniziare la programmazione.

Pseudocodice e PDL

Pseudocodice è un termine vago che indica un modo qualsiasi di definire con precisione quello che deve fare una routine nella normale lingua di tutti i giorni. A volte le routine vengono scritte prima in pseudocodice, che quindi viene sostituto con il codice effettivo, trasformandosi nel commento al codice stesso.
Il PDL (Program Design Language) è una versione formale di pseudocodice che è stata creata dalla società Caine, Farber & Gordon negli anni '70.
Per descrivere una routine si dovrebbe utilizzare lo pseudocodice, o PDL, anziché descrivere il modo in cui il linguaggio di programmazione specifico implementa l'idea. Una tecnica efficace consiste nello scrivere pseudocodice sempre più specifico finché diventa quasi automatica la traduzione in codice di programma.

Sfortunatamente, spesso non ci si può permettere il lusso di passare il tempo a pianificare e a progettare (anche se il mio consiglio è di trovare comunque del tempo per farlo). Qualsiasi cosa comporta dei compromessi. Un truismo sul software è che è possibile progettare un programma mirando a uno di questi tre aspetti: la velocità, il consumo di risorse e la facilità di manutenzione.

La logica che sta dietro questo vecchio modo di dire è che il codice veramente ottimizzato in funzione della velocità o del consumo di risorse a volte risulta realmente contorto. Con la progettazione moderna dei programmi, in particolare in un ambiente come quello di VB, che in ogni caso non eccellerà mai in velocità o in consumo di risorse, l'ago della bilancia si sposta verso la facilità di manutenzione.

Vi sono eserciti di programmatori che si occupano della manutenzione dei programmi legacy dei mainframe. Anche i programmi di Visual Basic 6 scritti oggi potrebbero aver bisogno di manutenzione nei prossimi anni (si potrebbe addirittura sostenere che i problemi di manutenzione dei programmi di VB sono di gran lunga superiori a quelli delle applicazioni mainframe a causa del linguaggio molto più ricco, flessibile ed estensibile). Se non si considera la manutenzione come un aspetto molto importante di qualsiasi progetto, negli anni a venire si avranno molti problemi. Nello stesso modo in cui il "problema dell'anno 2000" ha riportato al lavoro eserciti di programmatori di Cobol, chi scrive oggi programmi di VB senza tener conto della manutenzione può essere sicuro di mantenere a lungo il posto di lavoro.

Un programma semplice da mantenere deve essere chiaro e facile da leggere (ovviamente per chi capisce la programmazione e il linguaggio in cui è stato scritto). Non sono quasi necessari commenti, perché si commenta da solo: dando solo un'occhiata alle routine si capisce cosa fanno e perché lo fanno. I nomi delle variabili sono intuitivi. Al posto di numeri oscuri per le costanti si utilizzano nomi. Alla concisione si preferisce la *chiarezza,* funzionale.

Spesso i programmatori che si occupano della creazione di codice lavorano molto per trovare un modo più semplice per fare qualcosa. In generale, questa è un'ottima abitudine. Infatti quasi sempre, quando si cerca di risolvere un problema in modo brutale, si perde solamente tempo. Esiste un cliché in questo settore che dice che un programmatore pigro è un buon programmatore. Tuttavia, se il modo più semplice di fare qualcosa non è chiaro, forse non è il modo migliore. In altre parole, in tutti i casi si dovrebbe ricercare un modo intelligente o ingegnoso di fare qualcosa, ma solo se lo si capisce realmente e se qualcuno con cui non si è mai discusso in •recedenza dell'argomento (vale dire un programmatore che si occupa di manutenzione) è in grado di capirlo.

Progettazione dell'architettura delle applicazioni

L'organizzazione di livello superiore di un progetto, vale a dire la sua architettura, deve essere pianificata attentamente. Ogni routine deve avere uno scopo chiara-

mente definito e distinto. Poiché un modulo è un gruppo di routine, ognuno di essi deve presentare un fondamento razionale e coerente per il raggruppamento.
Inoltre spesso ha senso suddividere le applicazioni in modo strutturale: una parte può essere utilizzata per incapsulare la verifica delle "regole del business" o "regole dell'azienda" un'altra per accedere a un database e un'ultima parte per interagire con gli utenti e il loro input. Questa idea piuttosto intuitiva a volte è chiamata archi lettura "multitiered" o "multilivello".
L'approccio adottato dall'Information Technology Group (ITG) di Microsoft all'interno della stessa società utilizza quattro livelli architettonici:

- interfaccia utente (UI)
- dati
- transazioni
- accesso esterno

In questo modello, il livello dell'interfaccia utente contiene tutto il codice necessario per rispondere all'interazione dell'utente, e nient'altro. Questo livello è l'unico che dovrebbe fare riferimento ai form, ai controlli e così via. Include tutto il codice per la gestione degli eventi e gestisce la visualizzazione, rispondendo ai cambiamenti di stato delle finestre e inizializzando le richieste degli utenti.
Il livello dei dati fornisce tutti i dati necessari per la visualizzazione al livello della UI e tutti i dati al livello delle transazioni. Il livello dei dati si occupa delle operazioni locali di inizializzazione e dell'organizzazione, della formattazione e dell'ordinamento dei dati in entrata e in uscita. Inoltre in questo livello dovrebbe trovarsi l'attuazione delle regole dell'azienda.
Il livello delle transazioni usa quello dei dati come una specie di buffer di informazioni ed è responsabile di controllare il funzionamento del livello di accesso all'esterno, che è quello che comunica effettivamente con le sorgenti di dati esterni.
Nonostante questo modello sia utilizzato per applicazioni client/server di dimensioni considerevoli, i principi generali possono essere applicati a programmi di qualsiasi dimensione. Si deve cercare di raggruppare il codice simile, separandolo in base alla funzionalità intesa, e di generalizzare il codice per facilitarne il riuso.

Convenzioni per l'attribuzione dei nomi

La chiarezza dei nomi è estremamente imFortante. Il modo più semplice per sapere cosa contiene una variabile è di leggerne il nome, se questo è stato definito in modo appropriato. In modo simile, la prima indicazione di cosa fa una routine è data dal suo nome. Per questo motivo il nome delle routine dovrebbe essere un verbo attivo: una routine fa sempre qualcosa, ad esempio AnalizzaInput (Parselnput), Confronta (Do Compare), ImpostaValorePredefinito (Set Default) e così via. I nomi delle variabili, se attribuiti correttamente, indicano ai lettori cosa può essere assegnato ad esse. È preferibile pensare in modo concettuale e definire i nomi delle variabili in base a ciò che rappresentano e non in base a un processo di programmazione: Contatore, per esempio, non è un nome particolarmente utile.

La convenzione ungherese per l'attribuzione dei nomi

Charles Simonyi, un ungherese, ha inventato una convenzione per l'attribuzione di nomi di variabili che viene applicata comunemente alla programmazione in C. I nomi ungheresi sono composti da un tipo di base (minuscolo) seguito da un prefisso e da un qualificatore.

E pratica comune utilizzare lettere singole per le variabili di interi (ad esempio / o *X) eie* lettere maiuscole per rendere più semplice la comprensione dei nomi delle variabili e delle procedure. Ad esempio, Sub CalcolaMaggiore( ) è più semplice da leggere di Sub Calcolamaggiore().
Microsoft Consulting Services suggerisce di utilizzare convenzioni per l'attribuzione dei nomi che si basano vagamente sui nomi ungheresi (si veda il riquadro precedente). Lo scopo principale delle proposte di Microsoft è di definire un prefisso di tre lettere per i riferimenti ai controlli e ai form che indichi la natura dell'oggetto a cui si fa riferimento (ad esempio, txtManifesto per una casella di testo e frmMostraInventario per un form) e un prefisso per i nomi delle variabili composto da lettere che ne indicano il tipo e l'ambito (ad esempio gbStretch per una variabile globale booleana). Seguendo queste convenzioni si ha il vantaggio che nella finestra Properties gli oggetti dello stesso tipo sono tutti raggnippati, in quanto iniziano con lo stesso prefisso.
Come forse avete notato, gli esempi riFortati nel libro non sono sempre coerenti. Pur confermando quanto ho detto precedentemente, vale a dire che la *chiarezza, è* ciò che imForta realmente, quando si lavora in un gruppo diventa più imFortante attenersi a un'unica convenzione per l'attribuzione dei nomi. Personalmente tendo a seguire i suggerimenti della convenzione ungherese modificata da Microsoft per i nomi degli oggetti, ma non per le variabili. Nei progetti molto semplici a volte non mi preoccupo neanche degli oggetti e accetto i nomi predefiniti, ad esempio *Listi, List2,* se sono sufficientemente chiari.

Proprietà e metodi personalizzati dei form

È semplice implementare proprietà e metodi personalizzati, vale a dire definiti dall'utente. Il metodo o le proprietà create utilizzando la parola chiave Public sono imponibili per gli altri moduli nel progetto. Il vantaggio che si ottiene aggiungendo proprietà e metodi personalizzati a un form è che si incapsulano i form: tutto il codice di cui necessita il form si trova in un unico posto che è parte del modulo del form.

Aggiunta di metodi personalizzati

Supponete ad esempio di voler aggiungere il metodo .Center a un form per centarlo sullo schermo (naturalmente si può ottenere lo stesso effetto all'avvio di un'applicazione impostando la proprietà StartUpPosition del form). È possibile

aggiungere al form una procedura Public Sub che è una variante della procedura CenterForm generale utilizzata precedentemente (si veda il modulo APICode.Ba nel Capitolo 11):

```
Public Sub Center()
   Me.Move (Screen.Width - Me.Width) \ 2, _
      (Screen.Height - Me.Height) \ 2
End Sub
```

Questa procedura può essere chiamata utilizzando l'operatore punto (.), come qualsiasi altro metodo. Per esempio, se il nome delform è Form1 *e ilprogetto inizia da* Sub Mairi, *il seguente codiceprima carica e visualizza ilform, quindi chiama il metodopersonalizzato che lo centra sullo schermo (il codice di esempio è salvato nel CD-ROMallegato al libro con il nome Custom. Vbp):*

```
Public Sub Main()
   Form1.Show
   Form1.Center
End Sub
```

Aggiunta di proprietà personalizzate

È possibile aggiungere una nuova proprietà a un form semplicemente dichiarando una variabile pubblica nel modulo del form. Ad esempio:

```
Public MyProperty As String
```

Quindi si possono assegnare valori a questa proprietà personalizzata (che può anche restituire valori) all'esterno del modulo del form. In Sub Main, MyProperty potrebbe essere impostata nel seguente modo:

```
Form1.MyProperty = "Elbereth Gilthoniel!"
```

È possibile accedere al valore in MyProperty da qualsiasi punto, seguendo le normali regole sull'ambito delle proprietà. Da Form1 non è necessario nessun identificatore di modulo:

```
Private Sub Form_Load()
   Me.Caption = MyProperty
End Sub
```

Le variabili dei form e i metodi personalizzati possono essere utilizzati senza che sia necessario caricare il form nella memoria.
Le procedure abbinate **Property Get** e **Property Let** costituiscono un altro modo per aggiungere proprietà personalizzate a un form. **Property Get** è utilizzata per restituire un valore, **Property Let** per assegnarlo. In questo modo si può eseguire il codice all'interno delle procedure **Property Get** e **Property Let.** Rispettando le regole della buona pratica di programmazione, si possono utilizzare **Property Get** e **Property Let** per nascondere parzialmente o interamente i dati di un oggetto ed esForre l'oggetto solo per l'accesso selettivo.

Creare proprietà di sola lettura è semplice: è sufficiente fornire una procedura Property Get senza la corrispondente procedura Property Let e modificare la variabile protetta privata a cui fa riferimento la procedura all'interno del codice. Ad esempio, si supponga di avere una variabile chiamata MyWealth che registra quanti soldi si hanno. Si può dichiarare MyWealth come variabile privata a livello del modulo del form:

```
Private MyWealth As Double
```

L'accesso di sola lettura al valore di MyWealth può essere implementato per mezzo di una procedura Property Get:

```
Property Get Wealth() As Double
   Wealth = MyWealth
EndProperty
```

Ora si può utilizzare la proprietà *Wealth* per visualizzare il valore corrente di MyWealth:

```
Private Sub Command1_Click()
   Label1 = Format(Wealth, "Currency")
End Sub
```

Allo stesso tempo, il codice interno protetto può incrementare o decrementare il valore di MyWealth. In questo esempio piuttosto banale, un controllo timer aggiunge $100.00 a MyWealth ogni volta che viene attivato:

```
Private Sub Timer1_Timer()
   MyWealth = MyWealth + 100#
End Sub
```

In termini più formali, se si utilizzano le coppie di procedure Property Get e Property Let, Get agisce come una funzione: l'ultimo parametro di Let equivale al valore restituito della procedura Get abbinata.
Ad esempio, se si dichiara una variabile privata a livello di modulo

```
Private EntityName As String
```

è possibile creare procedure Property Get e Property Let che includono codice di esecuzione:

```
Puublic Property Get Elvishness() As String
    If EntityName = "Frodo" Then
      Elvishness = "Hobbit"
   Else
      Elvishness = "Elf"
   End if
  EndProperty

Public Property Let Elvishness(vNewValue As String)
     vNewValue = "Saruman" Then
     MsgBox "Non è un valore valido per un Elfo!"
   Else
```

```
        EntityName = vNewValue
    End If
End Property
```

Nel progetto di esempio si accede alla proprietà Elvishness nell'evento Click d' un pulsante di comando:

```
Private Sub Command2_Click()
    Command2.Caption = Form1.Elvishness
End Sub
```

Se si assegna il valore Saruman alla proprietà Elvishness in Sub Main:

```
Form1.Elvishness = "Saruman"
```

viene visualizzato il messaggio "Non è un valore valido per un Elfo!" e il valore di EntityName non viene modificato. Se a Elvishness si assegna il valore *Frodo*

```
Form1.Elvishness = "Frodo"
```

il valore restituito della procedura Property Get Elvishness è Hobbit. Quando a Elvishness viene assegnato qualsiasi altro valore, ad esempio

```
Form1.Elvishness = "Legolas"
```

il valore restituito è Elf.
La creazione di proprietà e metodi personalizzati può essere molto potente e può aiutare a definire in modo appropriato l'accesso alle variabili a livello di form. Purtroppo non è possibile aggiungere proprietà personalizzate ai form di VB che durante la progettazione verranno inclusi nella finestra *Properties.*

Generare eventi personalizzati

Gli eventi personalizzati possono essere generati all'interno di un modulo di classe (le operazioni con i moduli di classe sono discusse nel Capitolo 14). Il funzionamento è piuttosto semplice: all'interno del modulo di classe si dichiara un evento (quello utilizzato in questo esempio è chiamato Fired):

```
Public Event Fired()
```

L'evento è generato (o "fatto scattare") utilizzando la dichiarazione RaiseEvent, che in questo esempio è inclusa in una funzione di classe chiamata beenFired:

```
Public Function beenFired()
    RaiseEvent Fired
End Function
```

È necessario che sia chiaro che siete voi gli unici responsabili della generazione di qualsiasi evento personalizzato, perché questo compito non viene svolto automaticamente.

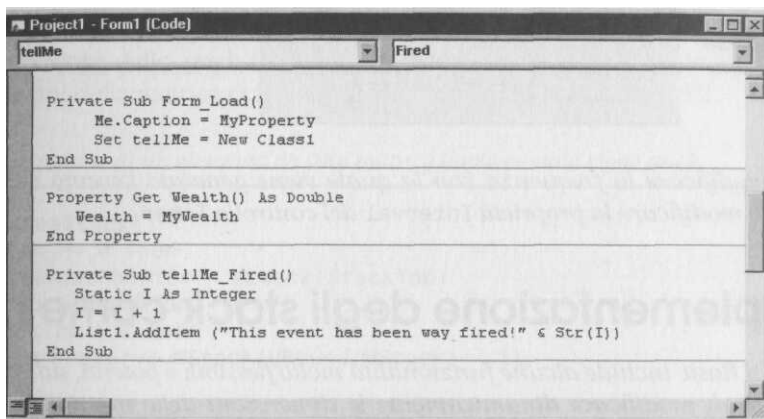
In questo esempio viene chiamato il metodo beenFired del modulo di classe (che general'eventoFired) dall'evento Timer di Form1, che viene aggiunto a MyWealth. Ogni volta che MyWealth aumenta, viene chiamato beenFired e viene generato **Fired**:

```
Private Sub Timer1_Timer()
   MyWealth = MyWealth + 100#
   tellMe.beenFired
End Sub
```

Per fare in modo che VB aggiunga una struttura per la gestione degli eventi a Form1, all'inizio del forni è necessario dichiarare un'istanza privata della classe utilizzando la parola chiave WithEvents:

```
Private WithEvents tellMe As Class1
```

Se si fa clic sull'elenco *Objects* di Form1, si vede un oggetto tellMe. Come mostrato nella Figura 13.1, nell'elenco *Procedures* è presente un evento Fired corrispondente.

Figura 13.1 *È possibile aggiungere al gestore di eventi un evento generato in un modulo di classe, dichiarando un 'istanza del modulo di classe con la parola chiave WithEvents.*

Prima di poter utilizzare un metodo di classe come beenFired, *si deve creare un'istanza della classe (oltre a dichiarare la classe). Se la classe genera eventi personalizzati, si deve utilizzare la dichiarazione* Set. *Ad esempio è possibile creare l'istanza* tellMe *di Classi nell'evento* Load *di* Form1:

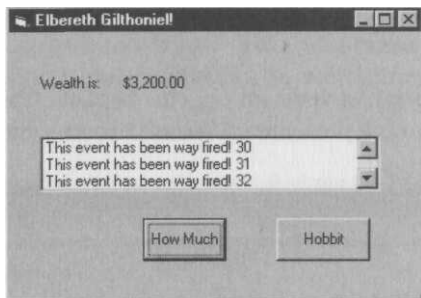
```
Private Sub Form_Load()

   Set tellMe = New Class1
End Sub
```

Poi è possibile aggiungere all'evento Fired *della classe* tellMe *un meccanismo per la visualizzazione. Nella realtà probabilmente all'evento si aggiungerebbe del codice da elaborare (dopo tutto questo è il motivo per cui si crea un evento personalizzato). Per i principianti tuttavia in questo modo perlomeno è possibile verìficare che l'evento è stato generato.*

Questo codice aggiunge una riga a una casella di riepilogo ogni volta che viene generato l'evento tellMe, vale a dire quando l'evento Timer incrementa MyWealth. Una variabile statica incrementa il contatore dell'evento se l'applicazione è in esecuzione-

```
Private Sub tellMe_Fired()
   Static I As Integer
   1 = 1 + 1
   List1.AddItem ("Questo evento è scattato!" & Str(I))
End Sub
```

Nella Figura 13.2 è mostrata la casella di riepilogo dopo che sono stati generati diversi eventi tellMe.

Figura 13.2
// codice inserito nel gestore di evento personalizzato viene eseguito ogni volta che viene generato l'evento personalizzato.

Per modificare la frequenza con la quale viene generato l'evento tellMe, *è sufficiente modificare la proprietà* Interval *del controllo* Timer.

Implementazione degli stock come motrici

Visual Basic include alcune funzionalità molto flessibili e potenti, ad esempio la possibilità di modificare dinamicamente le dimensioni delle matrici (array) mantenendone i valori. Il seguente esempio, salvato nel CD-ROM con il nome Mouse. Vbp, mostra come si utilizza una matrice dinamica per implementare uno stack.

Uno stack è una struttura di dati con la caratteristica che ogni nuovo elemento che vi viene aggiunto diventa l'elemento successivo a cui si accede dallo stack (in base alla sua natura, uno stack non è una struttura ad accesso casuale; solo l'elemento superiore è disponibile per il programma).
Il cursore del mouse, chiamato anche "puntatore", cambia forma in modo casuale. Per memorizzare il valore del cursore corrente utilizza uno stack, permettendo all'utente di scorrere all'indietro lo stack fino ad arrivare al cursore precedente. Non vi sono limitazioni per quanto concerne le dimensioni di questo stack e il numero di cursori che possono essere mantenuti in memoria, ad eccezione del fatto che lo stack è implementato in una matrice di interi (se 32.676 cursori non fossero sufficienti, sarebbe comunque possibile dichiarare MStack come una matrice di valori long!). Di seguito sono riFortate le dichiarazione di modulo per MStack, la matrice dello stack, e di StackTop, la variabile che tiene in memoria l'elemento corrente nella matrice (quello in cima allo stack):

```
Option Explicit
Dim MStackO As Integer 'Stack del mouse
Dim StackTop As Integer 'Cima dello stack
```

La procedura PushStack aggiunge il cursore corrente nella parte superiore dello stack e, se necessario, modifica le dimensioni della matrice dello stack, come mostrato nel Listato 13.1.

Listato 13-1 *Aggiunta di un elemento a una matrice implementata come stack.*

```
Private Sub PushStack(NewMouseValue As Integer)
   If StackTop = UBound(MStack) Then
      ReDim Preserve MStack(UBound(MStack) + 1)
   End If
   StackTop = StackTop + 1
   MStack(StackTop) = Screen.MousePointer
   Screen.MousePointer = NewMouseValue
End Sub
```

PopStack funziona in modo inverso, sostituendo il cursore corrente con il valore nella parte superiore dello stack (il cursore precedente) e quindi diminuendo di uno le dimensioni della matrice dello stack, come mostrato nel Listato 13.2.

Listato 13.2 *Estrazione di un elemento da una matrice implementato come stack.*

```
Private Sub PopStack()
   If StackTop >= 0 Then
      Screen.MousePointer = MStack(StackTop)
      StackTop = StackTop - 1
      If StackTop > 0 Then
         ReDim Preserve MStack(UBound(MStack) - 1)
      End If
   End If
End Sub
```

Lo stack deve essere inizializzato con una dimensione:

```
Private Sub InitializeMouseO
   ReDim MStack(10)
   StackTop = -1
End Sub
```

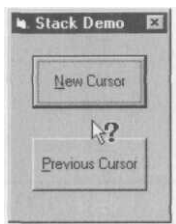
È possibile inserire una chiamata a InitializeMouse nell'evento Load del forni o in qualsiasi altro punto, in base alle necessità. Per scegliere un nuovo cursore a caso e inserire il valore del cursore corrente nella parte superiore dello stack, chiamare PushStack con un generatore di cursore casuale:

```
Private Sub cmdPush_Click()
   PushStack Rnd * 15
End Sub
```

I valori 0-15 in Visual Basic a 32 bit sono costanti MousePointer valide (si vedano gli argomenti "MousePointer Property" e "MousePointer Constants" nella Guida in linea di VB). L'utilizzo della funzione Rnd restituisce un valore MousePointer casuale.
Per ripristinare il cursore precedente, è sufficiente chiamare PopStack:

```
Private Sub cmdPop_Click()
   PopStack
End Sub
```

Il programma di esempio ora è completo e pronto per essere eseguito (si veda la Figura 13.3).

Figura 13.3 *Il programma Mouse Stack mostra come si utilizzano gli stack per memorizzare i valori del cursore.*

L'implementazione degli stack con le matrici dinamiche può essere un approccio corretto per creare in VB una struttura per l'accesso ai dati in situazioni in cui è richiesto l'accesso last-in-first-out (LIFO, l'ultimo che entra è il primo a uscire), anziché l'accesso casuale .

Interruzione dei cicli Do

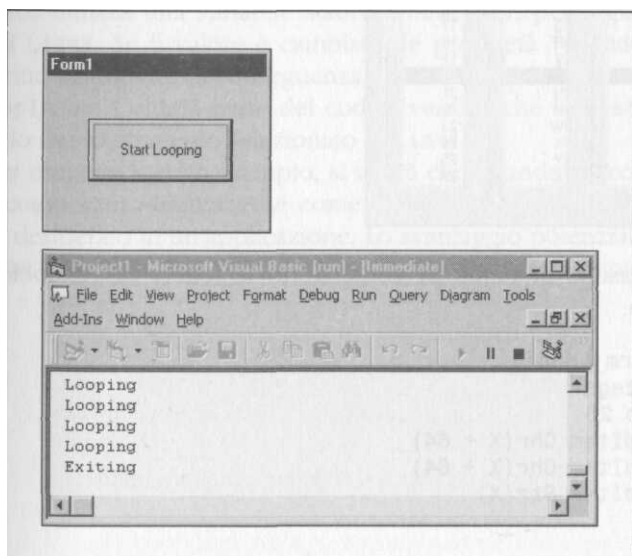
Un problema comune consiste nel permettere all'utente di interrompere i cicli. Per gli utenti è frustrante dover aspettare mentre il computer continua a lavorare senza aver la possibilità di interromperlo, se non premendo Ctrl+Alt+Canc. Ciò vale in particolar modo quando l'utente capisce di aver chiesto al computer di eseguire un'operazione errata (in Windows a 32 bit premendo Ctrl+Alt+Canc è perlomeno possibile terminare l'operazione corrente. Nelle versioni precedenti invece gli utenti si ritrovavano spesso a dover riavviare il computer).
Come parte di un'interfaccia "user-friendly", si deve permettere all'utente di interrompere un ciclo premendo un tasto o facendo clic su un pulsante del mouse (di solito il tasto Esc o il pulsante sinistro del mouse). Ovviamente è necessario fare in modo che in questo tipo di situazione i dati non siano corrotti a causa dell'interruzione di un calcolo.
Un modo comune per farlo consiste nel chiamare la funzione DoEvents all'interno di un ciclo lungo, la quale ottiene l'esecuzione in modo che il sistema possa elaborare altri eventi. Altre parti del programma possono quindi rilevare se sono stati premuti dei tasti specifici e fare in modo che il ciclo venga chiuso. Tuttavia questo è un processo tutt'altro che lineare e si possono incontrare diversi problemi per rientrare nel ciclo.
Un approccio migliore consiste nell'utilizzare l'API GetAsyncKeyState e controllare all'interno del ciclo per vedere se sono stati premuti tasti specifici. Il codice nel

Listato 13.3 interrompe un ciclo (che diversamente sarebbe infinito) quando l'utente preme il tasto Esc o fa clic sul pulsante sinistro del mouse:

Listato 13.3 *Interruzione di un ciclo Do.*

```
Option Explicit
Private Declare Function GetAsyncKeyState Lib "user32" _
    (ByVal vKey As Long) As Integer
Const VK_ESCAPE = &H1B
Const VK_LBUTTON = &H1

Private Sub Command1_Click()
    Do
        If GetAsyncKeyState(VK_ESCAPE) < 0 Or _
            GetAsyncKeyState(VK_LBUTTON) < 0 Then
            Debug.Print "Exiting"
            Exit Do
        End If
    'Qui va la logica del programma
    Loop
    Debug.Print "Looping"
End Sub
```

Gli argomenti costanti validi di GetAsyncKeyState ("VK" è l'abbreviazione di "Virtual Key") sono elencati nell'applicazione API Text Viewer sotto "Constants".
Nella Figura 13-4 è mostrata un'applicazione che implementa l'interruzione di un ciclo. Quando si sceglie il comando *Start Looping,* il ciclo inizia. Per ogni passaggio nel ciclo viene stampato un messaggio nel pannello *Immediate Debug* (per ulteriori informazioni sull'utilizzo del pannello *Immediate* si faccia riferimento al Capitolo 15). Quando l'utente preme il tasto Esc, il ciclo si arresta e viene stampato un messaggio.

Figura 13.4 *E sempre opportuno permettere agli utenti di uscire da un ciclo infinito.*

La logica del programma in questo esempio andrebbe all'interno del ciclo, dopo aver verificato se è stato premuto il tasto Esc.

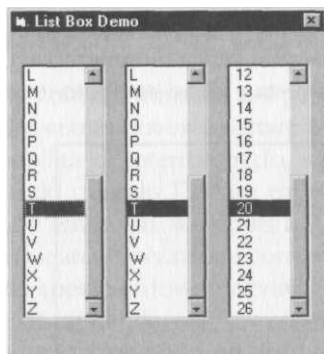
Gestione delle caselle di riepilogo

Un problema comune nelle applicazioni di VB è la gestione dei controlli casella di riepilogo. Di seguito sono presentate alcune tecniche che possono essere utilizzate nella maggior parte delle applicazioni in cui è necessario programmare questi controlli.

Registrazione di diverse caselle di riepilogo

Spesso vi devono essere due o più caselle di controllo che funzionano di concerto. Ad esempio, supponete che una casella riepilogo riForti i numeri degli elementi nell'inventario e un'altra il nome degli elementi. In questa situazione, quando si seleziona il numero di un elemento, è necessario un meccanismo che selezioni anche il nome correlato.

Il programma di esempio, salvato nel CD-ROM con il nome ListDemo.Vbp, mostra due tecniche per gestire questo problema comune della programmazione in VB. Con le due tecniche si ottengono funzionalità di visualizzazione leggermente diverse. Nella dimostrazione ho aggiunto a un form tre caselle di riepilogo, come mostrato nella Figura 13.5.

Figura 13.5
Gli eventi Click delle tre caselle di riepilogo sono connessi.

Le prime due caselle di riepilogo contengono le lettere dell'alfabeto, la terza i numeri da 1 a 26:

```
Private Sub Form_Load()
    Dim X As Integer
    For X = 1 To 26
        List1.AddItem Chr(X + 64)
        List2.AddItem Chr(X + 64)
        ListS.AddItem Str(X)
    Next X
End Sub
```

La prima tecnica utilizza la proprietà TopIndex della prima casella di riepilogo, List1 TopIndex, disponibile solo durante l'esecuzione, imposta o restituisce il valore indice del primo elemento nella casella di riepilogo (forse avete già familiarità con la proprietà ListIndex, che imposta o restituisce l'elemento correntemente se1ezionato). È possibile utilizzare la proprietà TopIndex in modo che, quando l'utente scorre la prima casella di riepilogo, le altre scorrano di conseguenza.
Per impostare questo tipo di situazione, si aggiunge un controllo timer al forni impostandone la proprietà .Enabled su True e la proprietà .Interval su un numero basso, per esempio 10.
Successivamente si aggiunge all'evento Timer del timer il codice incluso nel Listato 13.4:

Listato 13.4 *Sincronizzazione di caselle di riepilogo.*

```
Private Sub Timer1_Timer()
    Static PrevList1 As Integer
    Dim TopList1 As Integer
    TopList1 = List1.TopIndex
    If TopList1 <> PrevList1 Then
        List2.TopIndex = TopList1
        ListS.TopIndex = TopList1
        PrevList1 = TopList1
    End If
    If List1.ListIndex <> List2.ListIndex Or _
        List1.ListIndex <> List3.ListIndex Then
        List2.ListIndex = List1.ListIndex
        List3.ListIndex = List1.ListIndex
    End If
End Sub
```

Questo codice utilizza una variabile statica, PrevListl, per registrare il valore di TopIndex di Listi. Se il valore è cambiato, le proprietà TopIndex di List2 e di List3 vengono aggiornate di conseguenza e a PrevListl viene assegnato il nuovo valore di TopIndex. L'ultima parte del codice verifica che in List2 e in ListS sia selezionato lo stesso elemento selezionato in List1.
Se si prova a eseguire questo esempio, si vedrà che quando si scorre List1, List2 e List3 si comFortano esattamente come List1. Questo potrebbe essere il comportamento desiderato in un'applicazione. Lo svantaggio potenziale è che gli utenti possono agire solamente su Listi e non sulle altre caselle di riepilogo.

La proprietà Interval di Timer

L'unità di misura utilizzata dalla proprietà Interval del timer è il millesimo di secondo. Se Interval è impostata su 1.000, il timer viene attivato (cioè viene attivato il suo evento Timer) ogni secondo. Se Interval è impostata su 10, il timer è attivato 100 volte al secondo. Quando il timer viene utilizzato per aggiornare una visualizzazione, come nel programma di dimostrazione, in realtà non è molto imFortante quale numero si seleziona per la proprietà Interval, se è sufficientemente basso da non causare "sobbalzi" che possono essere notati durante il funzionamento del programma.

Per vedere un altro approccio, impostate la proprietà Enabled di Timer1 su False (in questo caso non ci si deve preoccupare dell'evento Timer di Timer1), quindi aggiungete al form la seguente procedura:

```
Private Sub SetList(c As Control)
   Static PrevList As Integer
   Dim TopList As Integer
   TopList = c.TopIndex
   If TopList <> PrevList Then
      List1 .TopIndex = TopList
      List2.TopIndex = TopList
      List3.TopIndex = TopList
      PrevList = TopList
   End If
   List1.List1ndex = c.ListIndex
   List2.List1ndex = c.ListIndex
   List3.List1ndex = c.ListIndex
End Sub
```

La logica di questa procedura è uguale a quella del gestore di eventi del timer. L'unica differenza è che viene chiamata dai gestori dell'evento Click di ogni casella di riepilogo:

```
Private Sub List1_Click()
   SetList List1
End Sub

Private Sub List2_Click()
   SetList List2
End Sub

Private Sub List3_Click()
   SetList List3
End Sub
```

In questo modo si connettono le tre caselle di riepilogo ogni volta che viene generato l'evento Click di una delle tre.
Una considerazione da fare in merito a questa seconda tecnica è che, nonostante le caselle di riepilogo siano sincronizzate ogni volta che viene generato un evento Click per una di esse (ad esempio quando si selezione una voce), scorrendole non

si genera l'evento Click. Le caselle di riepilogo connesse in questo modo rimangono disallineate finché l'utente sposta il cursore nell'area client di una delle caselle.

Evitare di eliminare involontariamente gli elementi nelle caselle di riepilogo

Il modo in cui si potrebbe istintivamente pensare di eliminare diversi elementi da una casella di riepilogo causa un errore di esecuzione. Per vedere cosa intendo dire, aggiungete una casella di riepilogo a un form e impostatene la proprietà MultiSelect *su* 1 -Simple. *In questo modo si permette all'utente di selezionare contemFora-neamente diverse voci. Successivamente aggiungete elementi alla casella di controllo (il codice è salvato si trova nel CD-ROM conIl nome Removing. Vbp).*

Si potrebbe pensare che il modo naturale per eliminare le voci selezionate da una casella di riepilogo sia di scorrere ciclicamente tutte le voci ed eliminare quelle selezionate, come nel seguente esempio:

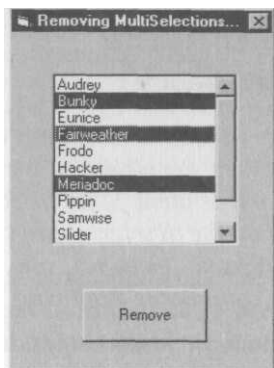
```
Private Sub cmdRemove_Click()
   Dim x As Integer
   For x = 0 To List1 .ListCount - 1
      If List1.Selected(x) Then
         List1.Removeltem x
      End If
   Next x
End Sub
```

Tuttavia, se si esegue questo codice quando vi sono diversi elementi selezionati, si ottiene l'errore di esecuzione 381 "Invalid property array index". Questo errore si verifica perché x, la variabile contatore, include un numero di elementi equivalente al numero massimo di elementi che erano presenti nella casella, anche se alcuni di essi sono stati eliminati durante la procedura. In altre parole, nonostante ListCount venga decrementato quando viene rimosso un elemento dalla casella di riepilogo, ciò non avviene per x. Ad un certo momento, pertanto, x sarà più grande del numero di elementi nell'elenco. La soluzione consiste nel convertire il ciclo For. . . Next in un ciclo Do While con un contatore interno, come mostrato nel Listato 13-5:

Stato 13.5 *Il modo corretto per eliminare diverse voci da una casella di riepilogo.*

```
Private Sub cmdRemove_Click()
   Dim x As Integer
   X = 0
   Do While x < List1.ListCount
      If List1.Selected(x) Then
         List1.Removeltem x
      Else
         X = x + 1
      End If
   Loop
End Sub
```

Quando sono selezionate diverse voci, come mostrato nella Figura 13.6, utilizzando questo metodo è possibile cancellarle.

Figura 13.6 *// modo corretto per eliminare diverse voci da una casella di riepilogo con un ciclo Do.*

Copiare negli Appunti le voci selezionate in una casella di riepilogo

Per terminare la discussione sulle caselle di riepilogo, di seguito viene mostrato come copiare negli Appunti le voci selezionate in una casella di riepilogo (e come incollarle in un controllo RichTextBox)*.// codice è salvato nel CD-ROM con il nome Clip.Vbp. Queste semplici operazioni sono indipendenti dai controlli, cioè si possono utilizzare le stesse tecniche per copiare e incollare da e in qualsiasi controllo.*

Per ulteriori informazioni sull'oggetto Clipboard e sui suoi metodi, fare riferimento alla Guida in linea di VB. Le costanti degli Appunti si trovano nell'argomento "Clipboard Object Constants"; le si può trovare anche utilizzando PObject Browser. Per copiare negli Appunti diversi oggetti da una casella di riepilogo, si utilizza il metodo SetText dell'oggetto Clipboard, come mostrato nel Listato 13.6:

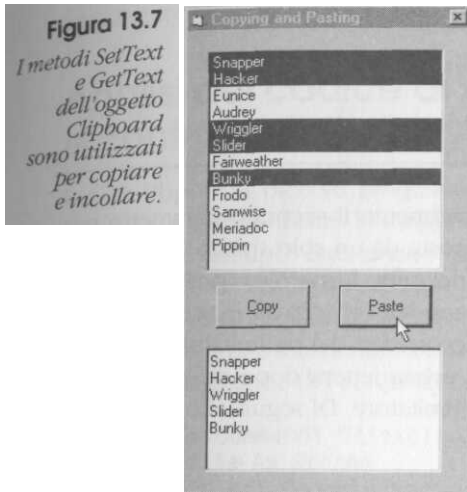
Listato 13.6 *Copia di diverse voci negli Appunti.*

```
Private Sub cmdCopy_Click()
    Dim ClipStr As String, I As Integer
    Clipboard.Clear
    ClipStr = ""
    For I = 0 To List1.ListCount - 1
        If List1.Selected(I) Then
            ClipStr = ClipStr & List1.List(I) + vbCrLf
        End If
    Next I
    Clipboard.SetText ClipStr
End Sub
```

Per incollare il contenuto degli Appunti in un controllo RichTextBox, si utilizza il metodo GetText dell'oggetto Clipboard (con un argomento costante appropriato):

```
Private Sub cmdPaste_Click()
  RichTextBox1.Text = ""
  RichTextBox1.Text = Clipboard.GetText(vbCFText)
End Sub
```

Nella Figura 13.7 viene mostrato come copiare diverse voci selezionate da una sella di riepilogo negli Appunti e dagli Appunti in una casella Rich Text.

Figura 13.7 *I metodi SetText e GetText dell'oggetto Clipboard sono utilizzati per copiare e incollare.*

Manipolazione delle stringhe

Come hanno imparato a proprie spese i programmatori più esperti, la corretta manipolazione delle stringhe è una parte estremamente imFortante dell'arte, o della scienza, della programmazione. Infatti, come mi ha detto un amico, alzando gli occhi dal monitor pieno di finestre di codice: "In un certo senso, tutta la programmazione può essere ridotta alla manipolazione delle stringhe".

Visual Basic offre un ambiente estremamente ricco per la manipolazione delle stringhe, con numerose funzioni integrate. Inoltre è molto semplice lavorare con il tipo i dati stringa nativo di VB. In questo paragrafo vengono presentate alcune tecniche necesarie per manipolare le stringhe in VB, che possono essere facilmente utilizzate in molte situazioni di programmazione reali.

Iniziare le parole in una stringa con la lettera maiuscola

Il Listato 13.7 contiene una funzione che cambia in maiuscola la prima lettera di ogni parola in una stringa.

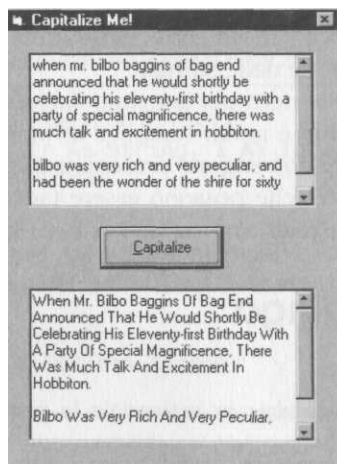
Listato 13.7 *Cambiare in maiuscola la prima lettera delle parole in una stringa.*

```
Private Function CapFirstLetter(InString As String, _
   DeLim As String)As String
   Dim PosDel As Integer
   Mid(InString, 1, 1) = UCase(Mid(InString, 1, 1))
   PosDel = InStr(InString, DeLim)
   While PosDel <> 0
      Mid(InString, PosDel +1, 1) = UCase(Mid(InString, _
         PosDel + 1, 1))
      PosDel = InStr(PosDel + 1, InString, DeLim)
   Wend
   CapFirstLetter = InString
End Function
```

La funzione qui è stata generalizzata, ma normalmente il secondo parametro passato (DeLim) dovrebbe essere una stringa composta da un solo spazio (" "), perché questo è il carattere che di norma separa, o delimita, le parole. CapFirstLetter innanzitutto cambia in maiuscola la prima lettera della stringa di input, quindi utilizza la funzione InStr per trovare la prima occorrenza del delimitatore (uno spazio). Quando la trova, cambia in maiuscola la prima lettera dopo di essa e quindi passa a ricercare l'occorrenza successiva del delimitatore. Di seguito ho applicato la funzione CapFirstLetter a due caselle di testo:

```
Private Sub cmdCap_Click()
   Text2.Text = ""
   Text2.Text = CapFirstLetter(Text1.Text, " ")
End Sub
```

Questo progetto è salvato sul CD-ROM allegato al libro con il nome Cap1rst.Vbp. Come si può vedere nella Figura 13.8, il risultato è piuttosto soddisfacente.

Figura 13.8 *È possibile utilizzare le funzioni UCase, Mid e InStr per cambiare in maiuscola la prima lettera di ogni parola.*

Vi sono tre cose da notare: questa funzione può essere utilizzata con qualsiasi controllo che contiene del testo, ad esempio un controllo RichTextBox. Inoltre, poiche le parole sono delimitate da un carattere di spazio, la prima lettera delle parole su una nuova riga creata con il ritorno a capo automatico non viene cambiata in maiuscola, a meno che non si immetta uno spazio prima della parola,

Infine poiché il delimitatore è uno spazio, la seconda parola di due parole uniti con un trattino non viene cambiata in maiuscolo. Ovviamente non sarebbe difficili aggiungere del codie per gestire questi ultimi due casi.

Analisi del codice di Visual Basic e controllo della lunghezza delle righe

Length.Vbp è un piccolo programma che dimostra come si utilizzano le funziona Len e Trim per verificare la lunghezza delle righe (utilizza anche Val e Str per la conversione dei tipi). È da notare che viene mostrato anche come si può trattare un file di codice di Visuali Basic come un qualsiasi file di testo che può essere aperto analizzato e manipolato a piacere. La parte iniziale del codice utilizza il controllo CommonDialog per selezionare un file .Bas da controllare (nel Capitolo 7 viene fornita una spiegazione del controllo CommonDialog).

```
Private Sub Command1_Click()
   Dim Ourfile As String
   CommonDialog1.CancelError = True
   On Error GoTo ErrHandler
   CommonDialog1.DialogTitle = "Select BAS Module"
   CommonDialog1.Filter = "VB Code Modules (*.Bas)|*.Bas"
   CommonDialog1.Flags=cdlOFNFileMustExist
   CommonDialog1.ShowOpen
   On Error GoTo 0
   Ourfile = CommonDialog1.filename
   CheckLen Ourfile, Val(txtLen.Text)
   Exit Sub
ErrHandler:
'User Canceled
End Sub
```

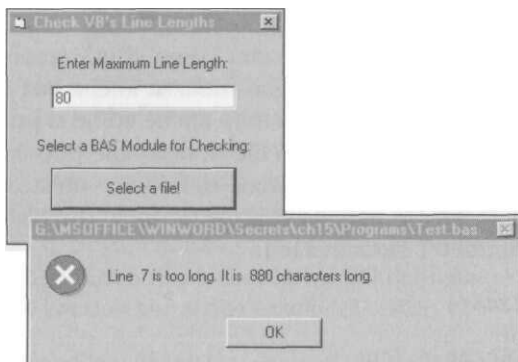
Dopo aver selezionato un file, viene chiamata la procedura CheckLen con il file e un valore per verificare la lunghezza delle righe, come mostrato nel Listato 1388:

Listato 13.8 *Controllo della lunghezza delle righe.*

```
Private Sub CheckLen(FN As String, CL As Integer)
  Dim Lin As String, Lcount As Integer
  Lcount = 1
  Open FN Fon Input As #1
  Do While Not EOF(1)
     Line Input #1, Lin
     If Len(Trim(Lin)) > CL Then
        MsgBox "Line " + Str(Lcount) + " is too long. It is " _
           + Str(Len(Trim(Lin))) + " characters long.", _
```

```
            vbCritical, FN
        End If
        Lcount = Lcount + 1
    Loop
    Close
End Sub
```

Il file selezionato viene aperto e assegnato, riga per riga, a una variabile, la cui lunghezza è confrontata con il parametro passato. Come mostrato nella Figura 13.9 Se la lunghezza è superiore al parametro, viene visualizzata una finestra di messaggio che include il numero e la lunghezza in caratteri della riga.

Figura 13.9 *I moduli di codice di Visual Basic possono essere analizzati come qualsiasi altro file di testo (nell'esempio viene controllata la presenza di righe troppo lunghe nei file .Bas).*

L'idea di analizzare e manipolare il codice di VB può essere ampliata e questa applicazione può fare molto di più. Non è un grande passaggio utilizzare le funzioni per la manipolazione delle stringhe per aggiungere o sottrarre caratteri di continuazione della riga, per creare rientri nel codice o per eseguire altri compiti automatizzati di elaborazione del codice.

Arrotondamento dei numeri

Il Listato 13.9 contiene una funzione generale che arrotonda i numeri a un numero arbitrario di posti decimali (per arrotondare a un intero si utilizza DecimalPlaces = 0):

Listato 13.9 *Arrotondare i numeri.*

```
Public Function RoundNumber(InNum As Variant, _
    DecimalPlaces As Integer) As Variant
    Dim Tmp As Double, DecShift As Long
    Tmp = CDbl(InNum)
    DecShift = 10 DecimalPlaces
    RoundNumber = (Fix((Tmp + 0.5 / DecShift) * DecShift)) / _
        DecShift
End Function
```

Ho chiamato la funzione da un forni con tre caselle di testo (si veda la Figura 13.10) nel seguente modo (il progetto è salvato nel CD-ROM allegato al libro con il nome RoundNum.Vbp):

```
Private Sub cmdRound_Click()
   txtResult.Text = RoundNumber(Val(txtNumber.Text), _
      Val(txtPlaces.Text))
End Sub
```

Figura 13.10
Per arrotondare i numeri si possono utilizzare le funzioni integrate di Visual Basic.

Si noti l'utilizzo della funzione Fix in RoundNumber. Come la funzione Int, anche Fix restituisce la parte intera di un numero, se questo non è negativo. In tal caso Fix restituisce il primo intero negativo maggiore o uguale al numero (mentre Int restituisce il primo intero negativo inferiore o uguale al numero).

Creazione di elenchi dei tipi di carattere

Spesso è necessario determinare quali tipi di carattere sono disponibili su un determinato sistema per la visualizzazione e la stampa. Il codice nel progetto Fonts.Vbp utilizza le proprietà Fonts e FontCount degli oggetti Printer e Screen per visualizzare i tipi di carattere disponibili per ogni oggetto (si veda la Figura 13.11), come mostrato nel Listato 13.10. Per determinare i tipi di carattere disponibili per entrambi gli oggetti, si può eseguire un ciclo in entrambi gli elenchi e aggiungere solo quelli comuni a entrambi.

Figura 13.11 *È possibile utilizzare gli oggetti Printer e Screen per visualizzare i tipi di caratteri disponibili.*

Listato 13.10 *Visualizzazione dei tipi di carattere.*

```
Private Sub Form_Load()
    DisplayFonts
End Sub

Private Sub DisplayFonts()
    Dim X As Integer
    For X = 0 To Printer.FontCount - 1
        lstPrinter.AddItem Printer.Fonts(X)
    Next X
    For X = 0 To Screen.FontCount - 1
        lstScreen.AddItem Screen.Fonts(X)
    Next X
End Sub
```

Per aggiungere un po' di interesse alla visualizzazione risultante (si veda la Figura 13.11), nell'evento Click di ogni casella di riepilogo è stato inserito del codice che imposta il tipo di carattere nella casella di testo sul valore della voce corrente:

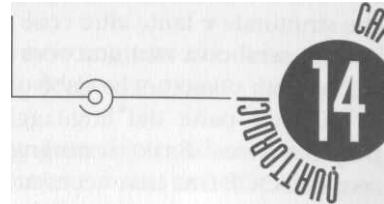
```
Private Sub lstPrinter_Click()
    lstPrinter.Font = lstPrinter.List(lstPrinter.ListIndex)
End Sub

Private Sub lstScreen_Click()
    lstScreen.Font = lstScreen.List(lstScreen.ListIndex)
End Sub
```

Poiché lstBox.List(lstBox.ListIndex) *restituisce sempre la voce* lstBox *corrente, questo codice fa sì che, ogni volta che viene generato l'evento* Click *correlato, ogni casella di riepilogo utilizzi la voce corrente come carattere di visualizzazione.*

Riepilogo

In questo capitolo si è passati dal generale allo specifico. Si è iniziato con una discussione generale della buona pratica di programmazione, della progettazione dell'architettura dei progetti e dell'attribuzione di nomi appropriati. Si è quindi passati ad argomenti più specifici.

- Si è appreso come aggiungere (e utilizzare) le proprietà e i metodi personalizzati dei form.
- Si è scoperto come impostare una proprietà personalizzata come proprietà predefinita di un oggetto quale un form.
- Si è appreso come generare un evento definito dall'utente in un modulo di classe.
- Si è appreso come rispondere agli eventi definiti dall'utente nei gestori di eventi dei form.
- Si è dimostrato come simulare uno stack per registrare i cambiamenti del cursore del mouse.
- Si è appreso come rispondere all'input dell'utente per interrompere un ciclo continuo.
- Si è scoperto come manipolare e controllare le caselle di riepilogo.
- Si è appreso come cambiare in maiuscolo la prima lettera di tutte le parole in una stringa di testo.
- Si è appreso come modificare il codice sorgente di Visual Basic memorizzato in un file su disco.
- Si è spiegato come controllare la lunghezza delle righe del testo memorizzato in un file.
- Si è appreso come arrotondare i numeri.
- Si è scoperto come creare elenchi di tutti i tipi di carattere disponibili per gli oggetti `Screen` e `Printer`.

VISUAL BASIC E L'OOP

- Analisi generale dell'OOP
- Visual Basic Versione 6 e l'OOP
- Le classi e i moduli di classe
- Gli oggetti e gli oggetti collezione
- L'oggetto Application
- L'utility Class Builder
- Estensione dei controlli esistenti

In questo capitolo verranno analizzati i concetti imFortanti dell'OOP (object-oriented programming, programmazione orientata agli oggetti) è verrà spiegato in che modo Visual Basic Versione 6 si integra nello schema OOP dell'universo della programmazione. Verrà anche dimostrato come sfruttare la potenza dell'OOP nei programmi di VB6.

Analisi generale dell'OOP

La teoria della programmazione orientata agli oggetti inizialmente è stata formulata come parte del processo di creazione di linguaggi destinati a simulare la vita reale. Ad esempio, Simula, uno dei primi linguaggi OOP e antenato di Smalltalk, era stato progettato dai norvegesi Ole-Johan Dahl e Krysten Nygaard nel 1967 come strunento per creare simulazioni (a proposito, Bjarn Stoustroup, il creatore di C++, ha riconosciuto di aver preso l'idea di come sono implementate le classi in C++ da Simula).

La simulazione sembra implicare naturalmente gruppi di cose: persone, molecole, azioni e così via. Il passo da "cosa" a "oggetto" è breve. Gli oggetti, come le persone, hanno caratteristiche interne (proprietà private). Hanno anche caratteristiche che presentano al mondo (proprietà pubbliche). È possibile dire agli oggetti cosa fare, e loro lo fanno, nel loro modo (rispettivamente metodi ed eventi). Inoltre molti oggetti condividono alcune caratteristiche comuni (ereditarietà).

Come suggerisce questa metafora, la programmazione OOP intensiva, al contrario della programmazione che utilizza solo alcuni concetti dell'OOP, quale l'incapsula-

mento degli oggetti, tende a funzionare meglio con sistemi che possono essere facilmente espressi utilizzando modelli organici. Questi sistemi sono, quasi per definizione, troppo complessi per essere compresi appieno in un modo convenzionale e procedurale e sono soggetti a continui cambiamenti. I sistemi delle previsioni meteorologiche e i mercati azionari possono essere due esempi. Come ha scritto Bruce Eckel dell'OOP: ". . .ho esercitato la programmazione procedurale, le tecniche strutturate e tante altre cose simili, ma non ha mai avuto granché senso per me, non mi sembrava mai una cosa completa . . . Poiché ora posso pensare in termini più potenti, posso anche risolvere problemi molto più complessi".
La maggior parte dei linguaggi di programmazione utilizzati oggi, inclusi C++, Delphi e Visual Basic, sono linguaggi OOP *ibridi,* cioè implementano alcuni concetti dell'OOP (ma non necessariamente tutti) e permettono anche la costruzione di programmi procedurali convenzionali. Java invece è un linguaggio OOP puro, cosa che costituisce contemForaneamente un punto di forza e uno svantaggio.
Come requisito minimo, per costruire programmi OOP un linguaggio deve fornire gli strumenti per il riutilizzo degli oggetti e del codice e la possibilità di creare all'interno del linguaggio nuovi oggetti estesi che si basano sugli oggetti esistenti.
Tra i linguaggi menzionati, Visual Basic è quello che storicamente ha avuto l'implementazione dell'OOP meno coerente, nonostante la Versione 5 e le successive contengano sufficienti funzionalità OOP da creare applicazioni in modo realmente orientato agli oggetti.

La cosa importante da capire è che Visual Basic 6 può essere utilizzato in modo OOP o meno, a seconda di cosa si preferisce. Le funzionalità dell'OOP nella Versione 6 che sono particolarmente imFortanti includono la possibilità di generare eventi personalizzati, di creare controlli ActiveX e di sottoclassificare il flusso di messaggi degli oggetti. Tutto ciò significa che è possibile creare programmi realmente orientati agli oggetti.

Naturalmente, come probabilmente sapete, VB ha caratteristiche particolari: per avere successo in questo ambiente, coloro che credono che "orientato agli oggetti" sia una frase per fanatici dovranno imparare ad esprimersi con Visual Basic e apprendere il linguaggio degli oggetti.

Incapsulamento

È possibile pensare all'*incapsulamento* come alla creazione di oggetti per mezzo dell'unione di dettagli dell'implementazione. In questo modo gli oggetti possono interagire chiamando i metodi degli altri oggetti, impostandone le proprietà e inviando messaggi. L'utilizzo degli oggetti incapsulati crea livelli di connessione tra le parti di un programma. Questo Forta in modo naturale a *nascondere* i dettagli dell'implementazione.
Gli oggetti nascondono i propri dettagli dell'implementazione permettendo l'accesso solo tramite routine controllate ("procedure di accesso"). Tutte le variabili e le procedure interne ("proprietà") hanno un ambito protetto e non è possibile accedervi o manipolarle dall'esterno dell'oggetto. Nascondendo i dettagli dell'implementazione si aiuta a creare codice modulare, riutilizzabile e di semplice manutenzione.

Con VB6 è semplice creare oggetti incapsulati complessi. Come detto in precedenza, i forni di VB sono oggetti con un'interfaccia visibile sullo schermo. Con i moduli di classe è semplice creare oggetti che non hanno componenti visibili e con le procedure `Property` è semplice creare routine di accesso agli oggetti.

Ereditarietà

Ereditarietà significa essere in grado di creare un nuovo oggetto che si basa su un oggetto esistente. Gli eventi, le proprietà e i metodi del nuovo oggetto derivano da quelli dell'oggetto originale.

L'ereditarietà permette di costruire classi in modo gerarchico. Ad esempio, *programmatore è* una sottoclasse di *persona che pensa* che è una sottoclasse di *persona.* In generale, la maggior parte degli oggetti della classe programmatore ha ereditato molte caratteristiche dell'oggetto persona, ad esempio gli occhi, le orecchie, i reni e così via. In modo simile, una casella di testo deriva da una classe generale che può essere definita come un controllo di finestra.

Sfortunatamente, VB permette un solo livello di ereditarietà in un progetto: `Dim X As New object`. Successivamente non è più possibile ereditare da X, cioè non è possibile creare classi di oggetti derivati definite dall'utente. Ad ogni modo è possibile ottenere alcuni vantaggi dell'ereditarietà aggiungendo un involucro agli oggetti esistenti e creando un nuovo oggetto che è una versione estesa dell'originale. Questo processo, a volte chiamato *delega,* è spiegato più avanti in questo capitolo nel paragrafo "Estensione di un controllo esistente".

In Visual Basic 6 è possibile creare i propri controlli ActiveX (si veda la Pane VI). Non vi è nulla che impedisce di creare un controllo sulla base di un controllo esistente, nel senso che il nuovo controllo eredita da quello precedente, ed è possibile ripetere questo processo tutte le volte che lo si desidera.

Quando si creano i propri controlli è particolarmente imFortante capire i concetti e l'implementazione degli oggetti e delle classi.

Polimorfismo

Polimorfismo significa che gli oggetti sanno quale azione devono eseguire, se viene loro inviato un messaggio che possono capire. La caratteristica più imFortante è che diversi oggetti possono eseguire azioni diverse se viene loro inviato lo stesso messaggio, perché implementano i metodi ereditati in modo diverso.

Se si chiede a dieci diversi oggetti programmatore di creare una procedura che fa *X* (il metodo *Programmatore. GoCodeX),* probabilmente si ottengono dieci diverse implementazioni di *X.* Se si hanno gli oggetti *Barca, Camion* e *Aeroplano,* questi sanno tutti cosa fare con una chiamata al metodo *SvoltaADestra.* Questo metodo produce un'azione diversa per ogni oggetto, a seconda dell'implementazione del metodo: le barche svoltano muovendo il timone, i camion girando le ruote e così via.

Nella maggior parte dei linguaggi OOP, il polimorfismo è implementato utilizzando l'ereditarietà. Ad esempio, le classi ipotetiche *Barca, Camion* e *Aeroplano* ereditano il metodo *SvoltaADestra* dal loro ipotetico genitore comune, la classe *Veicolo.*

Ogni classe ridefinisce il metodo *SvoltaADestra* ereditato dalla classe genitore Veicolo e aggiunge i propri dettagli di implementazione.
Visual Basic, invece, non implementa il polimorfismo per mezzo dell'ereditarietà, ma tramite diverse *interfacce* di controlli ActiveX esposte (l'interfaccia di un controllo è costituita dalle sue proprietà e dai suoi metodi esposti).

Per sfruttare appieno l'OOP in Visual Basic è necessario creare e utilizzare le gerarchie di controlli ActiveX.

Early binding e late binding

Early binding (associazione precoce) significa che il compilatore di VB sa quale oggetto viene chiamato e può controllare la libreria del tipo dell'oggetto per verificare se i membri chiamati sono presenti nell'oggetto. È possibile eseguire l'early binding degli oggetti dichiarando che utilizzano una classe specifica:

```
Dim X As Vehicle  'Early Binding
```

Late binding (associazione ritardata) significa che il compilatore di VB non è in grado di determinare di quale oggetto si tratta fino al momento dell'esecuzione. È necessario includere dell'ulteriore codice per verificare che i membri chiamati esistano realmente. Inoltre questo è un processo più lento, perché il controllo effettivo deve essere effettuato durante l'esecuzione. È possibile eseguire il late binding degli oggetti dichiarandoli `As Object`:

```
Dim X As Object   'Late Binding
```

Sistemi di messaggi

Nei programmi procedurali convenzionali, la logica è controllata per mezzo di dichiarazioni di controllo del flusso: `If`, `Do While` e così via. Ne consegue che l'intera logica del programma può essere concettualizzata e convertita in dichiarazioni concrete al momento della progettazione del programma, cosa che non sempre è vera. In particolare, spesso non è così nelle applicazioni progettate per creare il modello della dinamica dei cambiamenti nel mondo reale.
Nella programmazione orientata agli oggetti il controllo del flusso è determinato dai messaggi inviati agli oggetti. Si tratta di un modo più flessibile per simulare le condizioni del mondo reale. Gli oggetti rispondono ai messaggi che vengono loro inviati e possono inizializzare i messaggi per altri oggetti. In VB, per inviare un messaggio a un oggetto si chiama un metodo dell'oggetto.
La *sottoclassificazione (subclassing)* è una tecnica che permette di intercettare il flusso di messaggi, ad esempio quelli inviati a un forni o a un controllo, e di scrivere il proprio codice per estendere o modificare il comFortamento dell'oggetto che riceve i messaggi (l'espressione "sottoclassificazione" a volte è utilizzata anche per descrivere il processo di derivazione di una classe da un'altra).
Per sottoclassificare un flusso di messaggi è necessario utilizzare un *puntatore alla funzione,* che viene implementato utilizzando la parola chiave `AddressOf`. La fun-

zione definita dall'utente specificata quando si utilizza la parola chiave AddressOf è detta *callback (chiamata di ritorno) o funzione callback.* Nel Capitolo 11 sono riFortati alcuni esempi di implementazione delle funzioni chiamata di ritorno e della sottoclassificazione di finestre.

L'OOP in Visual Basic

Visual Basic, come la maggior parte dei programmi odierni, può essere utilizzato in modo più o meno orientato agli oggetti, a seconda della progettazione dell'applicazione e dello stile di sviluppo. Di seguito si vedrà come si può utilizzare VB6 come linguaggio di programmazione OOP.
Visual Basic supForta pienamente l'incapsulamento degli oggetti. I forni sono oggetti incapsulati con funzionalità di finestra; i moduli di classe sono oggetti incapsulati che non supFortano le finestre. È possibile utilizzare le procedure Property per nascondere in modo appropriato l'implementazione delle proprietà agli oggetti esterni.

I form come classi

Come probabilmente avete supposto, il form predefinito, Form1, creato dai progetti di Visual Basic in realtà è un'istanza di una classe. Per verificarlo, aprite un progetto Standard Exe e aggiungete il seguente codice all'evento Click diForm1:

```
Private Sub Form_Click()
   Dim newForm As New Form1
   newForm.Show
End Sub
```

Quando si esegue il progetto, se si fa clic sull'istanza predefinita di Form1 creata automaticamente da VB, viene visualizzata un'altra istanza identica di Form1. Una variabile oggetto dichiarata As New contiene Nothing finché la variabile non viene utilizzata, momento in cui VB crea un'istanza dell'oggetto che si basa sulla classe dichiarata. Nel seguente esempio

```
Dim newForm As New Form1
```

la classe è Form1.
Visual Basic crea una variabile oggetto globale nascosta per ogni classe di form. È come se VB avesse aggiunto al progetto la seguente riga invisibile di codice:

```
Public Form1 As New Form1
```

Quando si crea implicitamente un'istanza di Form1, definendo Form1 come oggetto di partenza o chiamando il metodo Show di Form1, in realtà si fa riferimento a questa variabile oggetto globale nascosta.

La collezione Forms, discussa più avanti in questo capitolo nel paragrafo "Gli oggetti collezione", tiene in memoria ogni classe di form nascosta in un progetto. È possibile utilizzare questa collezione per registrare e controllare i form in un progetto.

Fare riferimento agli oggetti

In VB è possibile derivare un nuovo oggetto da uno esistente utilizzando la parola chiave New. Questo rappresenta un livello di ereditarietà, ma non è sufficiente da permettere ai programmatori di sfruttare realmente i vantaggi che si ottengono lavorando che le gerarchie di classi estese. Ad ogni modo è possibile creare controlli ActiveX con tutti i livelli di ereditarietà che si desiderano.
La parola chiave As permette di dichiarare una variabile come un oggetto generale o in modo più specifico (per le differenza si veda il paragrafo "Early binding e late binding" precedentemente in questo capitolo). La parola chiave As ha un significato molto particolare se combinata con la parola chiave New, in quanto dichiara una variabile del tipo indicato e ne crea un'istanza nella memoria la prima volta in cui si fa riferimento ad essa.
Di conseguenza

```
Dim MyForm As Form1
```

è diverso da

```
Dim MyForm As New Form1
```

Nel primo caso viene creato un riferimento all'oggetto, ma in realtà non viene istanziato nessun oggetto nuovo. Più variabili oggetto possono fare riferimento allo stesso oggetto. Poiché queste variabili sono riferimenti all'oggetto, e non copie dello stesso, qualsiasi cambiamento dell'oggetto sottostante si riflette in tutte le variabili che fanno riferimento ad esso.
Nel secondo caso, quando si utilizza la parola chiave New nella dichiarazione Dim (o in una dichiarazione Set), viene creata un'istanza dell'oggetto. Questa nuova istanza non viene caricata in memoria finché non si fa riferimento ad essa o a un suo membro.

In questo capitolo vengono utilizzate molte volte le espressioni "istanziare" o "creare un'istanza". Entrambe hanno più o meno lo stesso significato di "creare".
Questa confusione nasce principalmente a causa del fatto che le classi devono essere istanziate esplicitamente prima di potervi fare riferimento:

```
Dim x As New clsShips
```

mentre i form possono essere istanziati ("creati") esplicitamente o anche implicitamente. La creazione esplicita dell'istanza di un form avviene esattamente come la creazione di un 'istanza di classe:

```
Dim X As New Form1
```

la creazione implicita dell'istanza di un forni avviene quando si carica un forni utilizzando il suo metodo Show o quando in VB viene iniziato un nuovo progetto con Form1. *In questo caso l'istanza creata è una copia di un oggetto forni interno, solo che alcuni dei meccanismi rimangano nascosti (si veda il paragrafo precedente "I form come classi").*

In un progetto predefinito impostato in modo da partire da Sub Mairi, se si cerca di assegnare un valore al riferimento a un oggetto:

```
Option Explicit
Dim x As Form1

Public Sub Main()
   x.Caption = "Lunga vita all'OOP!"
   x.Show
End Sub
```

si ottiene l'errore 91 "Object variable or With block variable not set", anche se nel progetto esiste un Form1. D'altra parte, se x viene utilizzato come una nuova istanza di Form1:

```
Option Explicit
Dim x As New Form1

Public Sub Main()
   x.Caption = "Lunga vita all'OOP!"
   x.Show
End Sub
```

viene caricata una copia di Form1 con la didascalia "Lunga vita all'OOP!", senza che vengano generati errori.

È possibile utilizzare la dichiarazione Set *per creare variabili che fanno riferimento agli oggetti. Ad esempio,* Set X = Form1, *assegna l'oggetto* Form1 *alla variabile* X. *Invece* Set X = New Form1, *assegna una nuova istanza di* Form1 *a X e la crea implicitamente. Generalmente la parola chiave* Dim *viene utilizzata con gli oggetti sui quali può essere eseguito l'early binding, mentre la dichiarazione* Set *è utilizzata con gli oggetti sui quali deve essere eseguito il late binding.*

Una funzionalità che fa di VB un linguaggio orientato agli oggetti è la possibilità di utilizzare oggetti integrati di VB. È possibile accedere alle proprietà e ai metodi di questi oggetti integrati. Esempi di oggetti sono l'oggetto App (Application), l'oggetto Screen, l'oggetto Printer e l'oggetto VBIDE (l'istanza corrente dell'IDE di VB, chiamata anche Visual Basic 6 Extensibility Library). Questi oggetti possono essere utilizzati dai programmatori in modo molto "orientato agli oggetti"
Nonostante possa essere molto utile accedere ad esempio alle proprietà dell'oggetto App (si vedano gli esempi più avanti in questo capitolo), ancora più imFortante per l'OOP risulta essere ActiveX. VB può essere utilizzato per creare oggetti ActiveX che espongono metodi a qualsiasi applicazione in grado di utilizzare metodi OLE di server esterni. Inoltre si possono utilizzare i metodi OLE esposti di applicazioni server all'interno di VB e anche creare controlli ActiveX.

L'integrazione di ActiveX in Windows e la possibilità di creare applicazioni server e controlli ActiveX in VB implicano che Windows, le reti private e il Web possono diventare una specie di ambiente di super-oggetti. In effetti ActiveX è un'implementazione generale dell'orientamento agli oggetti indipendente dal linguaggio e viene eseguito in diverse applicazioni.
Le applicazioni server di VB possono esForre metodi e proprietà ai client ActiveX e VB può utilizzare i metodi esposti di qualsiasi applicazione server ActiveX. I controlli ActiveX possono essere utilizzati privatamente all'interno di un progetto oppure possono essere registrati in un sistema ed essere utilizzati in altri progetti di VB6 o in qualsiasi ambiente di sviluppo in grado di fornire un contenitore per i controlli ActiveX. I controlli ActiveX creati in VB possono essere impiegati nel Web utilizzando Internet Explorer come contenitore.
Come è già stato accennato, è possibile manipolare VB in modo da definire un comFortamento più orientato agli oggetti di quello che avrebbe naturalmente, in particolare nelle aree dell'ereditarietà e del polimorfismo. È sufficiente utilizzare alcuni trucchi e, cosa più imFortante, definire degli standard di programmazione e attenersi ad essi. Nei paragrafi "Classi e moduli di classe" e "Estensione di un controllo esistente" viene mostrato come fare. Le tecniche apprese in questo capitolo si dimostreranno particolarmente utili per la creazione di controlli ActiveX.

Classi e moduli di classe

Una *classe* è un modello da cui viene creato un oggetto. In altre parole, una classe è un'idea o un costrutto, mentre l'oggetto correlato è un'implementazione, o istanza, o esemplare, dell'idea.
Per esempio, un form è un oggetto finestra. È possibile definire un nome di classe per ogni form (ad esempio Form1 o frmDemoClasser, più esplicativo) impostando la proprietà .Name dell'oggetto form. Ogni form, sia esso della classe Form1 o della classe frmClassDemo, si basa su una classe di form generica che Visual Basic implementa automaticamente. Questa classe generica di form ha tutte le proprietà (ad esempio Caption e BackColor) e i metodi (ad esempio Show e Line) familiari. Come sapete, è possibile espandere notevolmente la classe di form generica quando si creano classi di form personalizzate (e oggetti form che si basano su queste classi). In modo simile, un modulo di classe (un file .Cls) è un costrutto che non crea finestre utilizzato come modello per oggetti che sono istanze della classe. Le proprietà e i metodi di una classe (incluse le classi che si basano sul form) sono definiti *membri.* Nei moduli di classe è possibile avere qualsiasi proprietà e rnetodo desiderato quale implementazione del concetto di classe. I form e i controlli includono diversi eventi predefiniti, ad esempio il form Load e gli eventi Click. Al contrario, i moduli di classe hanno due soli eventi: Initialize e Terminate. È possibile generare gli eventi nei moduli di classe utilizzando la dichiarazione RaiseEvents. Questi eventi possono essere aggiunti alla struttura per la gestione degli eventi di un oggetto utilizzando la parola chiave WithEvents. Nel paragrafo "Generazione di eventi personalizzati" nel Capitolo 13 è riFortato un esempio di come si genera un evento personalizzato in un modulo di classe.

Eventi dei moduli di classe

L'evento Initialize di un modulo di classe viene generato quando un'applicazione crea un'istanza di una classe (l'evento Initialize di un forni viene generato nello stesso modo). Ciò avviene *indirettamente* facendo riferimento nel codice a una proprietà di un'istanza di una classe (o di un form). Per esempio, se si chiama un modulo di classe MyClass e gli si attribuisce una proprietà Public dichiarando una variabile pubblica Prop (come spiegato nel Capitolo 13), è possibile aggiungere del codice all'evento Initialize della classe per controllare quando viene generato:

```
Option Explicit
Public Prop As String

Private Sub Class_Initialize()
  MsgBox "I've been fired!"
End Sub
```

Per visualizzare la casella di messaggio "l've been fired!", bisogna fare riferimento a un'istanza di MyClass in qualsiasi altra parte del progetto, ad esempio nell'evento ClickdiForm1:

```
Private Sub Form_Click()
  Dim X As New MyClass
  X.Prop = "Howdy-doody!"
End Sub
```

L'evento Terminate di un modulo di classe (o di un form) viene generato quando vengono rimossi dalla memoria tutti i riferimenti a un'istanza della classe quando tutte le variabili che fanno riferimento all'oggetto sono impostate su Nothing o quando viene a mancare l'ambito dell'ultimo riferimento all'oggetto. Per esempio, aggiungete del codice all'evento Terminate della classe MyClass per visualizzare una finestra di messaggio:

La funzione CreateObject

Se si crea un'istanza di un oggetto di automazione OLE è possibile utilizzare un riferimento indiretto, se l'interfaccia dell'oggetto è stata aggiunta al progetto utilizzando la finestra di dialogo References. L'altra possibilità consiste nell'utilizzare direttamente la funzione CreateObject:

```
Dim X As Object
Set X = CreateObject("MyClass")
```

Per creare una classe che possa essere istanziata come oggetto di automazione OLE, è necessario impostare la proprietà Instancing del modulo di classe in modo che possa essere creato. Per ulteriori informazioni si veda la discussione di seguito e la Parte V del libro.

```
Option Explicit
Public Prop As String

Private Sub Class_Initialize()
  MsgBox "I've been fired!"
End Sub

Private Sub Class_Terminate()
  MsgBox Prop, vbInformatici-i, "Dylan Thomas: 1952"
End Sub
```

Se a Form1 si aggiunge una variabile a livello di form del tipo MyClass e indirettamente si istanzia la nuova classe nell'evento Load di Form1, quando l'istanza viene inizializzata e prima che sia visualizzata Form1 si ottiene la finestra di messaggio "I've been fired!" (si veda la Figura 14.1):

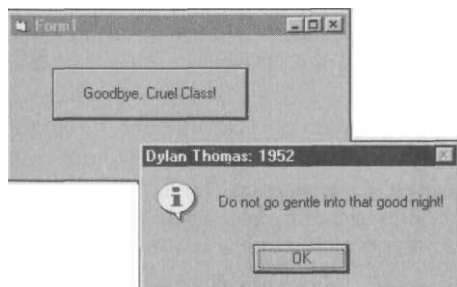
```
Option Explicit
Dim X As New MyClass

Private Sub Form_Load()
  X.Prop = "Do not go gentle into that good night!"
End Sub
```

Figura 14.1

Per attivare l'istanza dell'evento Terminate della classe, impostare la variabile che fa riferimento all'istanza su Nothing. Per esempio, nell'evento Click di un pulsante di comando,

```
Private Sub Command1_Click()
  Set X = Nothing
End Sub
```

impostando l'istanza della variabile di classe su Nothing si genera l'evento Terminate della classe, come mostrato nella Figura 14.2.

Figura 14.2
È possibile attivare l'istanza dell'evento Terminate della classe impostandone il riferimento alla variabile su Nothing.

Quando una variabile esce dal suo ambito

È possibile generare esplicitamente l'evento Terminate di una variabile di classe assegnando Nothing alla variabile. Inoltre l'evento Terminate viene generato quando non esiste più ambito per l'ultimo riferimento all'istanza della classe.
Questo significa che, se si istanzia X come classe del tipo MyClass a livello globale (del progetto), X mantiene il proprio ambito finché il progetto rimane in esecuzione; se X viene dichiarato a livello di form, mantiene l'ambito finché il form è caricato in memoria (e viene generato quando il form viene scaricato); se X è dichiarato come locale a una procedura, l'evento Terminate viene generato quando termina la procedura. Nell'esempio precedente, se X fosse dichiarato nella procedura dell'evento Load del form anziché a livello di form, come mostrato di seguito:

```
Option Explicit
Private Sub Form_Load()
   Dim X As New MyClass
   X.Prop = "Do not go gentle into that good night!"
End Sub
```

gli eventi Initialize e Terminate sarebbero generati uno dopo l'altro, ma prima che venga caricato Form1.

Proprietà dei moduli di classe

I moduli di classe hanno una o due proprietà intrinseche, a seconda del tipo di progetto in cui si trova il modulo di classe, come spiegato di seguito. In base alle impostazioni predefinite vi sono solamente due eventi e una o due proprietà. Non vi è nessun metodo; pur essendo possibile definire tutti i metodi personalizzati che si desidera per una classe, non ve ne è nessuno già integrato. Si può pensare alle proprietà, ai metodi e agli eventi integrati come se appartenessero al genitore della classe; in questo senso tutti i form hanno un metodo Show perché la classe radice dei form, che si trova all'interno di VB, ha un metodo Show.
L'unica proprietà che hanno tutti i moduli di classe è Name, che significa il nome della classe. I moduli di classe possono avere anche una proprietà Instancing, a seconda del tipo di progetto di cui fanno parte, come mostrato nella Tabella 14.1. Questa proprietà è utilizzata per specificare se è possibile creare istanze di una classe all'esterno di un progetto, come spiegato più avanti.

Tabella 14.1 *Tipi di progetto e valori delle proprietà che creano istanze di moduli di classe.*

Tipo di progetto	Un modulo di classe in questo tipo di progetto ha una proprietà Instancing?	Valori possibili per la proprietà Instancing del modulo di classe?
Standard EXE	No	N/A
ActiveX EXE	Sì	1 - Private 2 - PublicNotCreatable

(continua)

Tabella 14.1 *Tipi di progetto e valori delle proprietà che creano istanze di moduli di classe.*

Tipo di progetto	Un modulo di classe in questo tipo di progetto ha una proprietà Instancing?	Valori possibili per la proprietà Instancing de! modulo di classe?
		3 - SingleUse 4 - GlobalSingleUse 5 - Multiuse 6 - GlobalMultiUse
ActiveX DLL	Sì	1 - Private 2 - PublicNotCreatable 5 - Multiuse 6 - GlobalMultiUse
Aggiunta	Sì	1 - Private 2 - PublicNotCreatable 5 - Multiuse 6 - GlobalMultiUse
Controllo ActiveX	Sì	1 - Private 2 - PublicNotCreatable 5 - Multiuse 6 - GlobalMultiUse
DataProject (nuovo tipo di progetto in VB6)	No	N/D
Applicazione IlS (nuovo tipo di progetto in VB6)	Sì	1 - Private 2 - PublicNotCreatable 5 - Multiuse 6 - GlobalMultiUse
Documento DLL ActiveX	Yes	1 - Private 2 - PublicNotCreatable 5 - Multiuse 6 - GlobalMultiUse
Documento EXE ActiveX	Yes	1 - Private 2 - PublicNotCreatable 3 - SingleUse 4 - GlobalSingleUse 5 - Multiuse 6-GlobalMultiUse
Applicazione DHTML (nuovo tipo di progetto in VB6)	Sì	1 - Private 2 - PublicNotCreatable 5 - Multiuse 6 - GlobalMultiUse

È possibile impostare le proprietà Name e Instancing durante la progettazione nella finestra Properties, come mostrato nella Figura 14.3 (se il tipo di progetto supporta la creazione di istanze di moduli di classe). La proprietà Instancing determina se è possibile creare istanze di una classe all'esterno del progetto corrente.

Figura 14.3 *È possibile utilizzare la finestra Properties per impostare proprietà intrinseche di un modulo di classe durante la progettazione.*

I moduli di classe che non possono essere creati esternamente, ad esempio quelli nei progetti di controlli ActiveX, devono essere creati utilizzando i meccanismi forniti esplicitamente nel progetto.

La Tabella 14.2 mostra il significato delle sei possibili impostazioni della proprietà Instancing di un modulo di classe.

Tabella 14.2 *Impostazioni della proprietà Instancing dei moduli di classe.*

Impostazione	Significato
1 - Private	Il modulo di classe è privato (ha ambito locale) rispetto al progetto. Non può essere creato esternamente. Alle altre applicazioni non è permesso accedere alle informazioni della libreria di tipi relative alla classe e creare istanze di essa. Gli oggetti privati possono essere utilizzati solo all'interno dell'applicazione o del componente.
2 - PublicNotCreatable	Non può essere creato esternamente, ma può essere utilizzato esternamente dopo che è stato creato dall'applicazione o dal componente..
3 - SingleUse	Permette ad altre applicazioni di creare oggetti sulla base della classe, ma ogni oggetto di questa classe creato da un client inizia una nuova istanza del server..
4 - GlobalSingleUse	Simile a *SingleUse,* ad eccezione del fatto che le proprietà e i metodi della classe possono essere chiamati come se fossero semplicemente funzioni globali.

(continua)

Tabella 14.2 *Impostazioni della proprietà Instancing dei moduli di classe, (continua)*

Impostazione	Significato
5 - Multiuse	Permette ad altre applicazioni di creare oggetti sulla base della classe. Ogni istanza dell'applicazione può fornire qualsiasi numero di oggetti creati in questo modo, indipendentemente da quante applicazioni li richiedono. Se non è in esecuzione quando viene creato l'oggetto della classe, il server viene avviato.
6 - GlobalMultiUse	Simile a *Multiuse,* con un'aggiunta: le proprietà e i metodi della classe possono essere chiamati come se fossero semplicemente funzioni globali.

Non è necessario creare prima esplicitamente un'istanza della classe, in quanto viene creata automaticamente.

// modo in cui una classe può essere creata e il suo ambito sono entrambi inclusi nella proprietà Instancing *(in VB4 e nelle versioni precedenti, l'ambito era controllato da una proprietà separata). Quando una classe può essere creata, è possibile utilizzare le tecniche descrìtte precedentemente in questo capitolo per creare istanze della classe da altre applicazioni. Questo significa creare l'istanza esplicitamente, utilizzando la funzione* CreateObj ect, *come mostrato di seguito:*

```
Set MyInstance = CreateObject("MyProject.MyClass")
```

In alternativa è possibile utilizzare implicitamente la dichiarazione Din con le parole chiave As New;

```
Dim MyInstance As New MyClass
```

Le proprietà dei moduli di classe ActiveX sono discusse ulteriormente nella Parte V del libro. Nel Capitolo 13 è stato dimostrato come si aggiungono le proprietà ai form. Per aggiungere le proprietà ai moduli di classe si utilizzano le stesse tecniche:

- Le proprietà a cui è possibile accedere dall'esterno del modulo possono essere dichiarate come variabili pubbliche a livello del modulo.
- Le proprietà che si intende utilizzare solo all'interno di un modulo sono dichiarate come variabili private a livello del modulo.

È possibile scrivere procedure Property Get e Property Let per permettere l'accesso all'esterno alle proprietà nelle quali viene eseguito dell'ulteriore codice, ad esempio per la validazione. È possibile utilizzare le procedure Property per proteggere le variabili interne e per implementare proprietà di sola lettura.

Naturalmente, dopo essere state aggiunte a un modulo di classe, le proprietà vengono lette o scritte nel modo normale utilizzando una variabile di istanza della classe, come mostrato nel seguente esempio:

```
X.MyProperty = "Rattlesnake"
EmployeeList(I) = X.MyProperty
```

Le procedure Property Set

Leprocedure Property Set *si comFortano esattamente come leprocedure* Property Let, *ad eccezione del fatto che le prime sono utilizzate per impostare un riferimento a un oggetto, mentre le seconde sono utilizzateper impostare un valore.*

Come con Property Let, se si desidera poter leggere una proprietà, oltre che scriverla, Property Set deve essere abbinata a una procedura Property Get. Le procedure Property Let eseguono il codice ogni volta che alla proprietà viene assegnato un valore, mentre le procedure Property Set eseguono il codice ogni volta che un utente imposta un riferimento a un oggetto.
Per esempio, la seguente procedura Property Set cambia la didascalia del form che viene passato come argomento della procedura:

```
Public Property Set ChangeForm(frm As Object)
   frm.Caption = "Bulgy Bears are marshals of the Lists!"
End Property
```

Di seguito viene mostrato come si può chiamare la procedura Property Set (si veda la Figura 14.4):

```
Option Explicit
Public X As New MyClass

Public Sub Main()
   Dim Z As New Form1
   Set X.ChangeForm = Z
   Z.Show
End Sub
```

Figura 14.4 Bulgy Bears are marshals of the Lists!

Si noti che Set X.ChangeForm = Z *crea un'istanza di* X; *la classe, e chiama anche la procedura* Property Set ChangeForm *utilizzando 2 (un'istanza di* Form1) *come argomento.*

I moduli di classe e i tipi definiti dall'utente

I moduli di classe di VB corrispondono in modo naturale ai tipi definiti dall'utente .UDT, User Defined Types). Entrambi sono utilizzati per raggnippare dati strutturati definiti dagli utenti. È difficile immaginare del codice strutturato orientato in modo procedurale complesso che non utilizzi in grande quantità gli UDT (e infatti un altro termine per i tipi definiti dall'utente è "struttura definita dall'utente"). Tuttavia gli UDT non vanno molto d'accordo con le funzionalità OOP di VB6. Per esempio non e possibile creare una collezione di variabili di un tipo definito dall'utente (le collezioni di oggetti sono discusse più avanti in questo capitolo). Naturalmente si possono fare molte cose con gli UDT, ad esempio manipolarli in array e includere puntatori a funzioni come valori UDT. Tuttavia, per mettere meglio in pratica il

modello di OOP di VB, ci si deve abituare a utilizzare i moduli di classe anziché gli UDT (le classi e i tipi definiti dall'utente sono incompatibili: è sufficiente per esempio cercare di definire un UDT pubblico in un form o in un modulo di classe per provocare un errore di esecuzione).
Le variabili che compongono le parti di un tipo definito dall'utente diventano le proprietà pubbliche di un modulo di classe. Per esempio, all'UDT:

```
Public Type TNarnianShip
    Captain As String
    CrewNum As Integer
    Destination As Variant
    WaterBarrels As Long
End Type
```

si può accedere:

```
Dim DawnTreader As TNarnianShip
DawnTreader.Captain = "Prince Caspian"
```

Utilizzando un modulo di classe, chiamato clsNarnianShip, anziché una struttura, si definiscono proprietà pubbliche:

```
Option Explicit
Public Captain As String
Public CrewNum As Integer
Public Destination As Variant
Public WaterBarrels As Long
```

Alle proprietà di un'istanza della classe si accede in modo normale:

```
Dim DawnTreader As New clsNarnianShip
DawnTreader.Captain = "Prince Caspian"
```

Una differenza imFortante tra gli UDTe le istanze di classe è che a quest'ultime si possono assegnare variabili varianti, che quindi possono essere utilizzate per fare riferimento alla classe:

```
Dim Q As Variant

Set Q = DawnTreader
Form1.Caption = Q.Captain
```

Gli oggetti collezione

Un *oggetto collezione* (o *oggetto insieme)* rappresenta un modo per fare riferimento a oggetti correlati o per manipolarli. In VB vi sono alcune collezioni integrate. Inoltre è possibile definire le proprie collezioni di istanze di classi. In senso metaforico, le collezioni definite dall'utente sono per le istanze di classe ciò che le matrici sono per gli UDT. Le collezioni predefinite di VB sono:

- Controls: tutti i controlli in un form

- Forms: tutti i form in un progetto
- Printers: tutte le stampanti disponibili

Inoltre vi sono altri oggetti che contengono collezioni predefinite. Ad esempio l'oggetto VBE è l'oggetto radice che contiene tutti gli altri oggetti e le altre collezioni incluse in Visual Basic per Applicazioni. È possibile utilizzare le collezioni dell'oggetto VBE per accedere a:

- progetti, utilizzando la collezione VBProjects
- Windows, utilizzando la collezione Windows
- finestre per la modifica del codice, utilizzando la collezione CodePanes
- barre di comandi, utilizzando la collezione CommandBars

Un altro esempio imFortante è l'oggetto VBIDE, che rappresenta un'istanza dell'ambiente IDE di Visual Basic, che include diverse collezioni utili per creare aggiunte (si veda il Capitolo 29).
I controlli possono essere composti da collezioni di oggetti. Ad esempio, il controllo ImageList contiene una collezione ListImage, mentre ImageCombo contiene una collezione Comboltems (si veda il Capitolo 8).
È possibile iterare nelle collezioni utilizzando la dichiarazione For Each...Next. In alternativa è possibile utilizzare la proprietà .Count della collezione per passare ciclicamente da un elemento all'altro di una collezione mediante l'indice dell'elemento, anche se questo processo può creare confusione perché molte collezioni predefinite sono a base zero (ad esempio, For x = 0 to Collection.Count - 1) per motivi di compatibilita con le versioni precedenti, mentre le collezioni definite dall'utente sono a base uno (For x = 1 to Collection.Count). Inoltre, se si aggiungono o si eliminano elementi durante l'iterazione, l'indice e il conteggio vengono modificati, causando un errore nel ciclo. In parole povere, spesso è preferibile utilizzare For Each...Next.
Le collezioni si espandono e si contraggono automaticamente; è possibile pensare ad esse come a matrici che si occupano da soli di modificare le dimensioni. È possibile accedere agli elementi in una collezione tramite l'indice, come menzionato, ma in questo caso è necessario prestare attenzione perché gli indici possono cambiare quando si aggiungono o si rimuovo elementi. È anche possibile individuare gli elementi tramite una chiave stringa, se ne è stata assegnata una quando l'elemento è stato aggiunto. Nonostante sia possibile controllare in quale punto della collezione viene aggiunto un elemento (e anche ordinare gli elementi di una collezione), in base alle impostazioni predefinite gli elementi vengono aggiunti alla fine delle collezione, che non viene ordinata. Le collezioni hanno tre metodi: Add, Item e Remove (si veda la Tabella 14.3).

Tabella 14.3 *Metodi degli oggetti collezione.*

Metodo	**Sintassi**
Add	*Collection.Add(elemento, chiave, prima, dopo)*
Item	Collection.*Item (indice)*
Remove	Collection.Remove *indice*

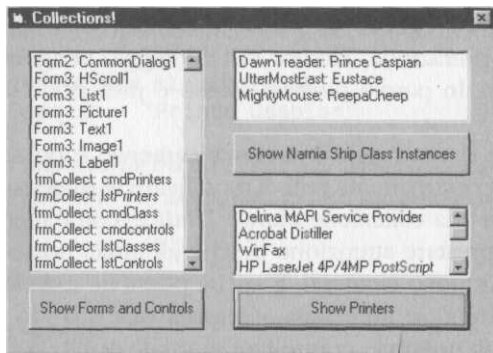
I metodi Add e Remove vengono utilizzati per aggiungere e rimuovere membri da una collezione. *Chiave, prima* e *dopo* sono argomenti opzionali. *Chiave* specifica una stringa unica che si può utilizzare al posto dell'indice per accedere a un membro specifico.
Si può specificare *prima* o *dopo,* ma non entrambi. L'argomento *prima* o *dopo* indica la posizione del membro in base al numero di indice o alla stringa di identificazione della chiave del membro rispettivamente successivo o precedente.
Il metodo Item è utilizzato per ritornare un membro specifico tramite il numero di indice o un valore chiave unico. Se il valore fornito per l'indice non corrisponde a un membro esistente della collezione, viene generato un errore di esecuzione.

Poiché Item è il metodo predefinito per gli oggetti collezione, le seguenti righe di codice sono equivalenti:

```
MsgBox   Forms(1).Caption
MsgBox    Forms.Item(1).Caption
```

Come si è detto, è preferibile utilizzare For Each. . .Next anziché la proprietà Count per accedere ai membri di una collezione. Molte collezioni predefinite vanno da 0 a Count-1 ; Forms(0) è il primo elemento della collezione di forni.

Utilizzando le collezioni di forni e di controlli è possibile visualizzare in una casella di riepilogo i nomi di tutti i form caricati e di tutti i controlli in ogni form, come mostrato nella Figura 14.5 (il codice di esempio è salvato nel CD-ROM in un progetto chiamato Collect. Vbp).

Figura 14.5
I molti usi dellecollezioni.

```
Private Sub cmdcontrols_Click()
    Dim obj As Object, ctl As Control
    lstControls.Clear
    For Each obj In Forms
        For Each ctl In obj.Controls
            lstControls.AddItem obj.Name & ": " & ctl.Name
        Next ctl
    Next obj
End Sub
```

Nello stesso modo è possibile visualizzare tutte le stampanti disponibili (si veda la Figura 14.5):

```
Private Sub cmdPrinters_Click()
    Dim prt As Printer
    For Each prt In Printers
        lstPrinters.AddItem prt.DeviceName
    Next prt
End Sub
```

Per scorrere e visualizzare una collezione definita dall'utente, è necessario innanzitutto definire la collezione. Utilizzando il modulo di classe clsNarniaShip, nel Listato 14.1 viene mostrato il codice (nel Sub Main del progetto di esempio) che definisce tre classi che si basano su di essa e viene creata una collezione delle classi:

Listato 14.1 *Creazione di una collezione di classi.*

```
Option Explicit
Public DawnTreader As New clsNarniaShip
Public UtterMostEast As New clsNarniaShip
Public MightyMouse As New ClsNarniaShip
Public Ship As New Collection

Public Sub Main()
    'Crea una collezione
    With Ship
        .Add DawnTreader
        .Add UtterMostEast
        .Add MightyMouse
    End With
    'Aggiungi valori di proprietà di classe e classi di istanza
    DawnTreader.Name = "DawnTreader"
    UtterMostEast.Name = "UtterMostEast"
    MightyMouse.Nane = "MightyMouse"
    DawnTreader.Captain = "Prince Caspian"
    UtterMostEast.Captain = "Eustace"
    MightyMouse.Captain = "ReepaCheep"
```

E possibile visualizzare i membri della collezione Ship in una casella di riepilogo (si veda la Figura 14.5) nel seguente modo:

```
Private Sub cmdClass_Click()
    Dim Member As Variant
    lstClasses.Clear
    For Each Member In Ship
        lstClasses.AddItem Member.Name + ": " + Member.Captain
    Next Member
End Sub
```

Questo esempio mostra come sia possibile utilizzare la collezione di form per scorrere tutti i form caricati, la collezione di controlli per scorrere i controlli in un form e la collezione delle stampanti per scorrere le stampanti disponibili. Infine è possibile scorrere i membri di una collezione definita dall'utente.

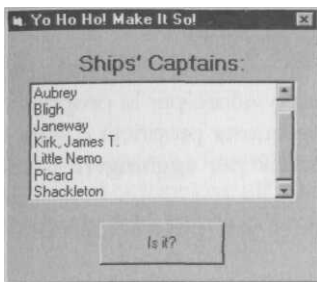
È una collezione?

Il Listato 14.2 mostra un'altra collezione, che si basa su una classe chiamata clsShip *con una proprietà* .CaptainName *(il progetto è salvato nel CD-ROM allegato al libro con il nome* IsItCol.Vbp):

Listato 14.2 *Un 'altra collezione.*

```
Option Explicit
Public Enterprise As New clsShip
Public Voyager As New clsShip
Public OldEnterprise As New clsShip
Public Endurance As New clsShip
Public Surprise As New clsShip
Public Peaquod As New clsShip
Public Submarine As New clsShip
Public Bounty As New clsShip
Public Ships As New Collection

Public Sub Main()
   Dim Member As Variant
   Enterprise.CaptainName = "Picard"
   Voyager.CaptainName = "Janeway"
   OldEnterprise.CaptainName = "Kirk, James T."
   Endurance.CaptainName = "Shackleton"
   Surprise.CaptainName = "Aubrey"
   Peaquod.CaptainName = "Ahab"
   Submarine.CaptainName = "Little Nemo"
   Bounty.CaptainName = "Bligh"
   With Ships
      .Add Enterprise
      .Add Voyager
      .Add OldEnterprise
      .Add Endurance
      .Add Surprise
      .Add Peaquod
      .Add Submarine
      .Add Bounty
   End With

   For Each Member In Ships
     Form1.lstCaps.AddItem Member.CaptainName
   Next Member
   Form1.Show
End Sub
```

Il codice carica la proprietà .CaptainName in una casella di riepilogo la cui proprietà Sorted è impostata su True (si veda la Figura 14.6). L'utilizzo delle caselle di riepilogo spesso è conveniente per ordinare gli elementi di una collezione. Naturalmente, per ordinare le collezioni sulla base dei loro indici è possibile scrivere del codice che utilizza i normali algoritmi di ordinamento.

Figura 14.6 *Una casella di riepilogo ordinata visualizza le proprietà CaptainName degli elementi nella collezione Ships.*

Come si può verificare se un oggetto è una collezione definita dall'utente anziché un altro tipo di oggetto simile a una collezione, ad esempio una casella di riepilogo? La seguente funzione, chiamata dal pulsante "Is it?" nel progetto IsItCol.Vbp, permette di scoprirlo verificando se una delle proprietà o dei metodi delle collezioni generano un errore:

```
Private Function IsItACollection(obj As Object) As Boolean
    Dim Var As Variant
    IsItACollection = True
    With obj
        On Error GoTo Fail
        .Add .Count
        Var = .Item(.Count)
        For Each Var In obj
        Next
        .Remove .Count
    End With
Exit Function
Fail:
IsItACollection = False
End Function
```

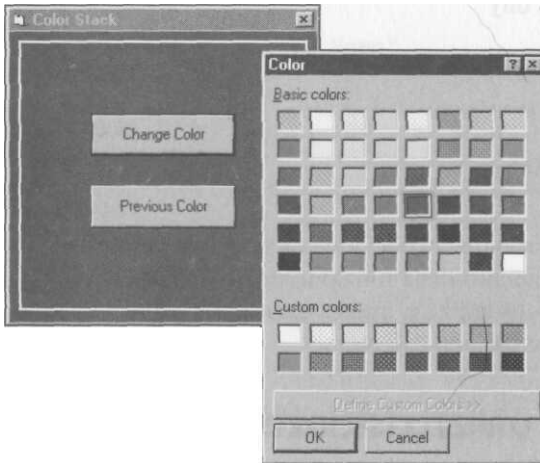
Non vi sono problemi a passare la collezione Ships come argomento della funzione; se si passasse come argomento della funzione una casella di riepilogo, la situazione sarebbe diversa. Questo esempio è interessante per vedere che cosa esattamente costituisce una collezione definita dall'utente.

Uno stock che utilizza istanze di classe e una collezione

Nel capitolo precedente è stato mostrato come si implementa uno stack utilizzando un array dinamico per registrare lo stato del cursore (si veda il paragrafo "Implementazione di uno stack come matrice"). In questo esempio viene dimostrato come si implementa uno stack utilizzando un'istanza di classe e una collezione. Nonostante anche in questo caso si possano registrare gli stati del cursore con lo stack, per cambiare ho utilizzato uno stack per registrare la proprietà .BackColor di un **form.** I metodi Push e Pop del modulo di classe dello stack in questo progetto (il

modulo di classe è Stack.Cls e il progetto è ColorStk.Vbp) possono essere utilizzati in qualsiasi situazione in cui si deve chiamare uno stack.
Per impostare questo progetto, ho utilizzato un controllo CommonDialog ShowColor per permettere all'utente di impostare un nuovo valore per la proprietà .BackColor per il forni (si veda la Figura 14.7). Dopo che questa proprietà è stata impostata, la procedura chiama il metodo Push di clsStack per aggiungere il vecchio colore alla fine dello stack:

```
Private Sub cmdChange_Click()
   Dim TmpColor As Long
   CommonDialog1.Flags = cdlCCRGBInit + _
      cdlCCPreventFullOpen
   CommonDialog1.CancelError = True
   On Error GoTo ErrHandle
   TmpColor = frmColorStack.BackColor
   CommonDialog1.Color = TmpColor
   CommonDialog1.ShowColor
   frmColorStack.BackColor = CommonDialog1.Color
   On Error GoTo 0
   x.Push TmpColor
   Exit Sub
ErrHandle:
   'Annullato - non fare nulla
End Sub
```

Figura *i4.7*
L'applicazione ColorStack mostra come implementare uno stack utilizzando una collezione e un 'istanza di una classe.

Assicurarsi di istanziare una variabile di classe sulla base del modulo di classe, in modo da poter chiamare i metodi e le proprietà della classe:

```
Dim x As New clsStack
```

Di seguito è riFortata la procedura che imposta la proprietà .BackColor *del form estraendo l'ultimo valore inserito nello stack:*

```
Private Sub cmdPrev_Click()
   If x.Count >= 1 Then
```

```
      frmColorStack.BackColor = x.Pop
   Else
      MsgBox "I'm back at the beginning!"
   End If
End Sub
```

Inoltre *utilizza* la proprietà .Count dell'istanza di classe in modo da non riutilizzare una seconda volta il colore che all'inizio si trovava in cima allo stack. Il Listato 14.3 mostra il contenuto del modulo di classe clsStack:

Listato 14.3 *Inserire ed estrarre elementi dallo stack di una collezione di classi.*

```
Option Explicit
Private ColorStack As New Collection

Public Sub Push(Var As Variant)
   ColorStack.Add Var
End Sub

Public Function Pop() As Variant
   With ColorStack
      If .Count Then
         Pop = ColorStack(.Count)
         .Remove .Count
      End If
   End With
End Function

Property Get Count() As Variant
   Count = ColorStack.Count
End Property
```

Come si può vedere, questa implementazione è estremamente semplice. La funzione Push aggiunge semplicemente l'argomento passato alla collezione; la funzione Pop restituisce l'elemento in cima alla collezione ColorStack (l'ultimo aggiunto) e quindi lo rimuove dalla collezione. La proprietà Count restituisce le dimensioni correnti della collezione, o dello stack.

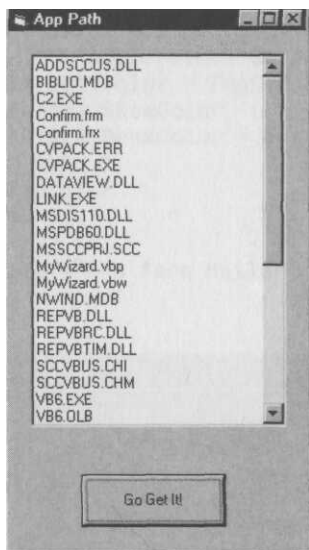
L'oggetto Application

E possibile accedere all'oggetto Application, chiamato oggetto App, utilizzando la parola chiave App. Nonostante l'oggetto App non abbia eventi o metodi, ha alcune proprietà che risultano utili per scoprire o per specificare informazioni sull'applicazione correntemente in esecuzione.
Nel Capitolo 11 è stato spiegato come si utilizzano le proprietà descrittive dell'oggetto App, ad esempio FileDescription e LegalCopyright, per riempire automaticamente le etichette nella finestra *About* di un progetto. Di seguito sono discusse le proprietà Path e PrevInstance dell'oggetto App.

App.Path

App.Path restituisce il percorso dell'applicazione corrente (il percorso della directory da cui è stata lanciata l'istanza corrente dell'applicazione). Ad esempio, il programma di esempio compilato AppPath.Exe copia tutti i file della directory in cui si trova in un controllo FileListBox (si veda la Figura 14.8). Sono necessarie solo due righe di codice per copiare i file dal percorso dell'applicazione nel controllo FileListBox e per copiare il percorso corrente nella didascalia del form:

```
File1.Path = App.Path
Form1.Caption = App.Path
```

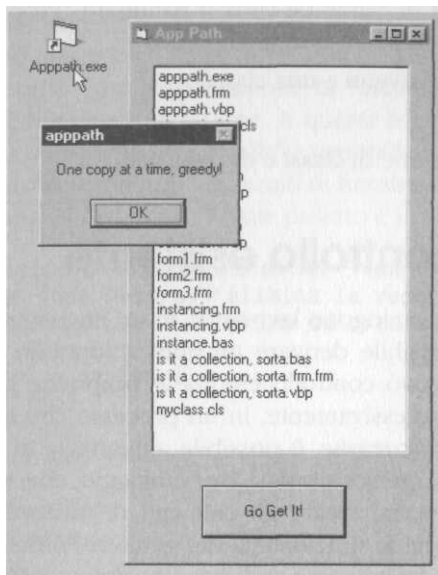
Figura 14.8 *È possibile utilizzare la proprietà Path dell'oggetto App per visualizzare tutti i file nel percorso dell'applicazione.*

App.PrevInstance

È possibile utilizzare App.PrevInstance per vedere se è in esecuzione un'istanza precedente di un'applicazione e per impedire, se lo si desidera, che vengano eseguite simultaneamente più copie di un'applicazione:

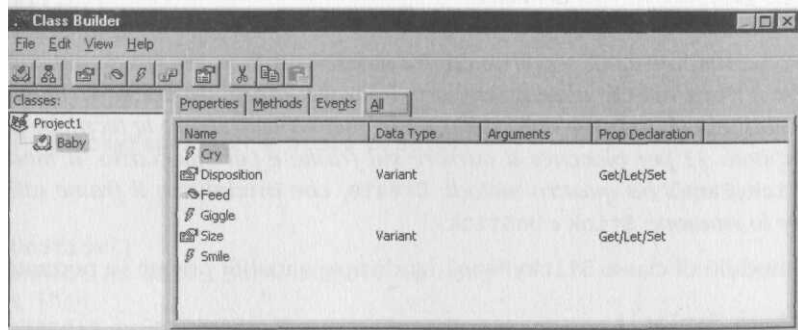
```
Private Sub Form_Load()
    If App.PrevInstance Then
       MsgBox "One copy at a time, greedy!"
       End
    Else
       MsgBox "No previous instance running!"
    End If
End Sub
```

Per eseguire questo codice si deve compilare il programma, diversamente la proprietà PrevInstance non calcola come istanze precedenti le copie di progettazione in esecuzione in diverse istanze della VBIDE. Dopo essere stata compilata, l'appli-

cazione AppPath utilizza la proprietà PrevInstance dell'oggetto App per rilevare se vi sono altre istanze in esecuzione, come mostrato nella Figura 14.9.

Figura 14.9
La proprietà PrevInstance dell'oggetto App può rilevare se è in esecuzione un 'istanza precedente dell'applicazione!

L'utility Class Builder

L'utility Class Builder è un'aggiunta di Visual Basic che aiuta a creare e a gestire le collezioni e le classi. Se è stata attivata l'aggiunta Class Builder selezionando la casella appropriata in Add-In Manager, nel menu *Add-Ins* viene aggiunta la voce *Class Builder.* Quando si sceglie questa voce, viene visualizzata una finestra di dialogo come quella mostrata nella Figura 14.10.

Figura 14.10
L'utilità Class Builder è un'aggiunta di VB che si può utilizzare per creare e gestire classi e collezioni.

E anche possibile avviare l'utilità Class Builder scegliendo VBClass *nella finestra di dialogo* Add Class Module.

Utilizzando Class Builder è possibile generare codice di programma che:

- crea una nuova classe
- imposta proprietà dei moduli di classe (si veda il paragrafo "Proprietà dei moduli di classe" precedentemente in questo capitolo)
- aggiunge proprietà, metodi ed eventi a una classe
- crea una nuova collezione
- definisce attributi per la creazione di classi e di collezioni

Estensione di un controllo esistente

Uno dei vantaggi principali che si ottengono lavorando in un linguaggio realmente orientato agli oggetti *è* che è possibile derivare un nuovo controllo da uno esistente. Questo significa che il nuovo controllo eredita le proprietà, gli eventi e i metodi del controllo esistente. Successivamente, in un processo che a volte viene chiamato *subclassing* o *sottoclassificazione,* è possibile aggiungere al nuovo controllo proprietà, eventi e metodi personalizzati. Un vantaggio che si ottiene in questo modo è che non è necessario creare gli elementi di un'applicazione dal nulla; il nuovo controllo include già le funzionalità del genitore, pronte per essere utilizzate. Nella Parte VI viene mostrato come si deriva un nuovo controllo da uno esistente. Nel frattempo, è semplice includere le funzionalità di un controllo esistente in un modulo di classe che quindi può essere esteso con proprietà e metodi definiti dall'utente, senza creare per intero il nuovo controllo. Questo processo è stato chiamato *delega,* nonostante *involucro* sia un termine maggiormente descrittivo, per che il vecchio controllo viene racchiuso con la nuova funzionalità nel modulo di classe. È imFortante notare tuttavia che gli involucri tuttavia non possono essere definiti senza che alla classe sia passata un'istanza del controllo che estende. Come con la maggior parte dei moduli di classe, il codice che incapsula una classe involucro è completamente riutilizzabile. Per utilizzare il controllo esteso più volte in un progetto, è sufficiente istanziare diverse copie della classe (e passare a ogni istanza un diverso controllo frame).

Probabilmente è più semplice capire questo concetto con un esempio. Delegate.Vbp, che si trova nel CD allegato al libro, è l'involucro di un controllo Frame *che ha due metodi estesi:* Stick *e* UnStick. *Questi metodi utilizzano le tecniche presentate nel Capitolo 11 per bloccare il cursore sul frame e per sbloccarlo. Il modulo di classe* StickyPanel *ha quattro metodi:* Create, *che inizializza il frame esteso;* Destroy, *che lo rimuove;* Stick *e* UnStick.

Il modulo di classe StickyPanel ha diverse variabili private (e pertanto interne):

```
Option Explicit
Private rPanel As Frame, fExist As Boolean, _
   FrameRect As RECT
```

rPanel serve come riferimento alla copia del controllo frame che StickyPanel utilizza. fExist verifica se la classe è già stata istanziata. FrameRect utilizza il tipo di

finestre RECT per rilevare i confini del cursore nel metodo .Stick (ho aggiunto a questo progetto il modulo APIDec.Bas del Capitolo 11, in modo da non dover inserire nuovamente l'API e le definizioni dei tipi).

Prima di poter utilizzare StickyFrame, è necessario crearlo. I controlli integrati di VB gestiscono automaticamente la propria creazione e distruzione; in questo caso è invece necessario farlo manualmente. La funzione Create restituisce un booleano che indica se è giunta a buon fine. A questa funzione deve essere passato un controllo Frame, nonostante sia possibile generalizzare questa definizione di classe in modo che funzioni con tutti i controlli di finestra. La funzione imposta il riferimento interno al frame sul controllo Frame passato e imposta il flag fExist su True.

```
Public Function Create(ptoPanel As Frame) As Boolean
   If fExist Then Destroy 'elimina le vecchie istanze
                          'prima di crearne una nuova
   On Error GoTo CreateError
   Set rPanel = ptoPanel
   fExist = True
   Create = True
   Exit Function
CreateError:
  MsgBox Error(Err)
  Create = False
End Function
```

Dall'altra parte del ciclo di vita del controllo esteso, Destroy imposta il flag fExist su False e rimuove dalla memoria il riferimento interno al frame:

```
Public Sub Destroy()
   Set rPanel = Nothing
   fExist = False
End Sub
```

Stick e UnStick verificano se StickyFrame è stato creato e quindi utilizzano la logica spiegata nel Capitolo 11:

```
Public Sub Stick()
   If fExist Then
      GetWindowRect rPanel.hwnd, FrameRect
      ClipCursor FrameRect
   Else
      MsgBox "StickyPanel non è stato creato!"
   End If
End Sub

Public Sub UnStick()
   Din ScreenRect As RECT, ScreenHandle As Long
   If fExist Then
      ScreenHandle = GetDesktopWindow 'Prende l'handle dello schermo
      GetWindowRect ScreenHandle, ScreenRect
      ClipCursor ScreenRect 'Ripristina il cursore
   Else
      MsgBox "StickyPanel non è stato creato!"
   End If
End Sub
```

Per utilizzare un controllo StickyFrame, creare prima un'istanza di una classe che si basa sul modulo di classe StickyPanel:

```
Dim X As New StickyPanel
```

Se sono necessari diversi controlli StickyFrame, *è sufficiente creare altre istanze di classe, come mostrato di seguito:*

```
Dim X As New StickyPanel, Dim Y As New StickyPanel, _
   Dim Z As New StickyPanel
```

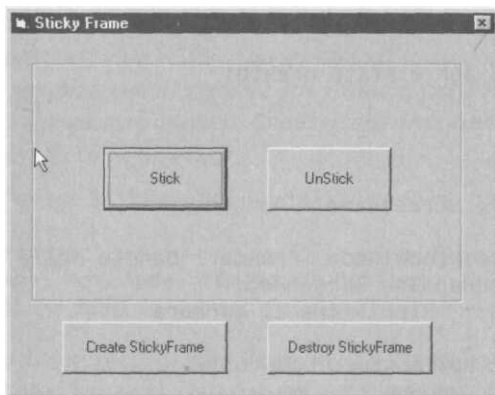
Poi si chiama il metodo Create della classe utilizzando come argomento il controllo Frame da estendere:

```
Private Sub cmdCreat_Click()
    Dim Retval As Boolean
    Retval = X.Create(Frame1)
    If Not Retval Then
       MsgBox "Impossibile creare il controllo StickyPanel!" _
    End If
End Sub
```

I metodi Destroy, Stick e UnStick si chiamano nel seguente modo:

```
Private Sub cmdDestroy_Click()
    X.Destroy
End Sub

Private Sub cmdStick_Click()
    X.Stick
End Sub

Private Sub cmdUnStick_Click()
    X.UnStick
End Sub
```

Come mostrato nell'applicazione di esempio (si veda la Figura 14.11), è stato esteso un controllo Frame facendo in modo che possa essere riutilizzato.

Figura 14.11 *Il progetto Delegation mostra come si estende un controllo esistente istanziando una classe con i metodi Create e Destroy.*

Riepilogo

Nonostante non includa un sistema di classi che permette di ereditare dalle classi esistenti, se non nel contesto dei controlli ActiveX, Visual Basic Versione 6 presenta tuttavia alcune funzionalità molto potenti. In realtà dipende dallo sviluppatore iniziare a pensare in un modo orientato agli oggetti e fare buon uso delle caratteristiche OOP di VB6.

- Si è appreso a scrivere codice in moduli di classe riutilizzabili che possono essere istanziati ripetutamente.
- Si è appreso come creare collezioni di oggetti definite dall'utente.
- È stato spiegato come si estendono i controlli esistenti con un modulo di classe involucro.
- Sono stati presentati i concetti di base dell'OOP, inclusi l'ereditarietà, l'incapsulamento, il polimorfismo e il sistema di messaggi.
- Si è appreso a istanziare e a creare classi e form.
- Sono stati discussi gli oggetti collezione, in particolare le collezioni Forms, Controls e Printers.
- Si è discusso dell'oggetto App.
- Sono state fornite informazioni sugli eventi, sulle proprietà e sui metodi delle classi.
- È stata spiegata l'utility Class Builder.
- Si è appreso a manipolare le collezioni definite dall'utente di oggetti classe.
- Si è scoperto come si utilizza un modulo di classe generico come stack.
- Si è appreso come creare un modulo di classe che estende le funzionalità di un controllo.

GESTIONE DEGLI ERRORI

- Comprendere i diversi tipi di errori
- Gli errori di sintassi e di compilazione
- Direttive per la verifica dei programmi
- Utilizzo delle dichiarazioni di errore di Visual Basic
- Programmazione con l'oggetto Err
- Generazione di un errore
- Generazione di un errore definito dall'utente
- Utilizzo degli strumenti di debugging di Visual Basic
- Le asserzioni

In questo capitolo sono discussi tre argomenti correlati: evitare i bachi nei programmi, includere codice per la gestione degli errori nel codice di VB e il debugging dei programmi che, sfortunatamente, includono bachi. Verranno descritti i diversi tipi di errore e verrà spiegato come gestire ognuno di essi.

Tipi di errori

Nei progetti di Visual Basic si possono incontrare tre tipi di errori:

- errori di sintassi e di compilazione
- errori di esecuzione
- errori di logica e di progettazione

Da dove vengono i "bachi"?

Si ritiene che il termine baco sia stato utilizzato per la prima volta negli anni '50 da] Contrammiraglio Grace Hopper, uno degli autori di COBOL, riferendosi a un vero insetto, un tipo di falena, che causava il malfunzionamento di uno dei primi mostri a valvole termoioniche. Oggi naturalmente per baco si intende qualsiasi errore o problema che causa il mal funzionamento di un programma. Per essere più precisi, l'utilizzo di questo termine per indicare un'anomalia o un problema in un processo o in una macchina risale a prima del ventesimo secolo. Si dice ad esempio che Thomas Edison abbia utilizzato questa espressione per fare riferimento a un problema con un'invenzione. Tuttavia l'Ammiraglio Hopper probabilmente è stata la prima persona a utilizzare questo termine facendo riferimento ai computer.

Gli errori di sintassi e di compilazione sono causati da codice che è stato creato in modo improprio. In altre parole, questi errori si verificano quando le istruzioni di un programma non soddisfano i requisiti della definizione formale del linguaggio (vale a dire che l'istruzione non può essere elaborata dal compilatore). È semplice individuare questi errori, a causa del numero limitato di istruzioni (e di conseguenza del numero limitato di errori possibili) e dell'eccellente controllo degli errori di sintassi incluso nell'ambiente di progettazione di Visual Basic (si veda il paragrafo "Errori di sintassi e di compilazione" più avanti in questo capitolo).
Può invece essere difficile risolvere gli errori di progettazione e logici (i classici, misteriosi "bachi nella macchina"), anche se sono di aiuto le numerose funzionalità di debugging integrate in Visual Basic. La cura migliore per gli errori di progettazione è la prevenzione, il che significa che si dovrebbe riservare sufficiente tempo per pianificare rigorosamente l'architettura del programma e le interfacce utente (si veda il Capitolo 13).
Ovviamente, a parte evitare di includere bachi, lo strumento migliore di debugging che ci possa essere è una mente acuta e analitica che possa eliminare una a una tutte le possibilità (nel caso peggiore, il debugging si richiama alla famosa frase di Sherlock Holmes: "Dopo aver eliminato l'impossibile, qualsiasi cosa, *per quanto improbabile,* deve essere la verità"). Gli strumenti moderni costituiscono però un aiuto; in un paragrafo alla fine di questo capitolo è spiegato come si utilizzano le numerose funzionalità di debugging di VB.
Si può certamente dire che chiunque abbia lavorato per un po' con VB può, con l'aiuto del controllo automatico della sintassi, produrre codice che viene compilato correttamente. Allo stesso modo, la maggior parte dei programmatori ha il proprio approccio per eseguire il debugging dei programmi che presentano problemi di logica.

Se si esegue un progetto nell'IDE di VB scegliendo il comando Start With Full Compile *(anziché semplicemente* Start) *dal menu* Run *permette di verificare tutto il codice nel progetto. Se si inizia senza la compilazione completa, viene verificato solo il codice che viene chiamato, e non tutto il codice incluso nel progetto. Oltre a scegliere il comando dal menu* Run, *per iniziare un progetto si può premere F5, mentre per iniziare un progetto con la compilazione completa si può premere Ctrl+F5.*

per esempio, può darsi che il codice nell'evento Click di un pulsante di comando contenga un errore. Se si avvia il programma senza la compilazione completa, non lo si scopre finché non si fa clic sul pulsante.
Gli errori di esecuzione sono potenzialmente un problema più imbarazzante, in quanto sono difficili da individuare durante la verifica di un programma. Possono invece presentarsi quando un cliente utilizza una funzione del programma che si trova molto all'interno della logica del programma e che viene utilizzata solo raramente. Anche le configurazioni hardware e i sistemi operativi "eccentrici" possono causare errori peculiari. I seguenti sono esempi di errori di esecuzione:

- una istruzione che cerca di effettuare una divisione per zero
- una istruzione che cerca di carica un file non presente
- una istruzione che cerca di accedere a un componente ActiveX non presente nel sistema
- il famoso errore di esecuzione numero 91: "Object variable or With block variable not set", spesso causato quando si fa riferimento a un oggetto, anziché istanziarlo, come nel seguente esempio,

  ```
  Dim x As Form1
  x.Caption = "blah"
  ```

  anziché

  ```
  Dim x As New Form1
  x.Caption = "blah"
  ```

- molti, molti altri!

Naturalmente alcuni errori di esecuzione sono così gravi da impedire la compilazione dell'intero programma (in questo caso si riceve un messaggio di errore). Solitamente sono simili a quelli di sintassi, ma sono troppo subdoli per poter essere rilevati dalla funzione *Auto Syntax Check.* Un esempio potrebbe essere l'errore numero 91 generato quando si cerca di assegnare un valore a un oggetto a cui è stato fatto riferimento (anziché un oggetto che è stato istanziato e che pertanto esiste). Un altro esempio è rappresentato dalla differenza tra il numero di parametri nella chiamata a una routine e il numero formale di argomenti della routine (in questo caso probabilmente si ottiene il messaggio "Argument not optional"). Questi "errori di sintassi" di esecuzione naturalmente devono essere risolti prima di poter continuare con il programma. Fortunatamente è abbastanza semplice eliminarli.
Un errore di esecuzione serio, vale a dire che ha passato il controllo del compilatore, è probabile che si presenti quando il cliente più imFortante carica per la prima volta il programma. Questo è un esempio tipico della legge di Murphy. Questi errori sono spesso causati da capricci della vita reale che semplicemente non si erano verificati precedentemente.

Visual Basic include alcuni strumenti utili per rilevare, o *intercettare,* gli errori di esecuzione. La libreria di esecuzione riconosce che si è verificato un errore e permette di intercettarlo e di intraprendere delle azioni correttive. La maggior parte di questo capitolo si concentra sull'utilizzo di questi strumenti per gestire gli errori di esecuzione nei programmi compilati, l'argomento di cui si preoccupano maggiormente gli sviluppatori seri.

Errori di sintassi e di compilazione

Gli errori di sintassi e di compilazione, detti in breve errori *di sintassi,* sono causati da codice creato in modo non appropriato, vale a dire da codice che non soddisfa i requisiti formali della definizione del linguaggio. Il *Language Reference* di Visual Basic, disponibile nella libreria MSDN inclusa in Visual Studio, include le definizioni formali di tutte le funzioni, parole chiavi e istruzioni di VB. Nel Capitolo 4 di questo libro è riFortato un riepilogo generale degli elementi del linguaggio di VB e di come è possibile riunirli.
Esempi di errori di sintassi sono l'invio di un numero errato di parametri a una procedura, l'inserimento di parole errate, l'omissione della punteggiatura nelle dichiarazioni di Visual Basic, la non corrispondenza di If e End If e di cicli in generale ed errori di digitazione delle variabili. Una causa comune di questi errori è l'omissione del carattere di continuazione della riga alla fine di una riga.
Questo tipo di errore è semplice da gestire, se si seguono alcune semplici tecniche. Per rilevare e risolvere velocemente gli errori di sintassi, si deve rendere obbligatoria la dichiarazione delle variabili nei progetti e attivare l'opzione *Auto Syntax Check* nella scheda *Editor* della finestra di dialogo *Options.*
Per rendere obbligatoria la dichiarazione delle variabili nei progetti si aggiunge la dichiarazione Option Explicit all'inizio di ogni modulo in un progetto. In alternativa è possibile scegliere *Tools\Options\Editor* e assicurarsi che sia attivata la casella *Require Variable Declaration.* In questo modo si inserisce automaticamente una istruzione Option Explicit all'inizio di ogni nuovo form o modulo (si noti che in questo modo l'istruzione Option Explicit non viene aggiunta ai moduli già inseriti, in quanto l'opzione *Require Variable Declaration* non agisce sui moduli esistenti).
Se si rendono obbligatorie le dichiarazioni delle variabili non è possibile utilizzare le dichiarazioni implicite (per ulteriori informazioni su questo argomento, si veda il Capitolo 4). Questo significa che non si possono eseguire i programmi che contengono variabili non dichiarate. Gli errori di digitazione nei nomi delle variabili vengono evidenziati immediatamente.
L'opzione *Auto Syntax Check* funziona in modo simile. La si imposta nella scheda *Editor* della finestra di dialogo *Options* del menu *Tools,* assicurandosi che sia attivata la casella di controllo *Auto Syntax Check* (è consigliabile lavorare sempre con questa impostazione attivata). In questo caso Visual Basic visualizza un messaggio di errore ed evidenzia il codice errato non appena nella finestra *Code* si immette un errore di sintassi. Di seguito sono riFortati due esempi di come funziona questa opzione.

// mio Syntax Error Text *è impostato in modo da visualizzare gli errori in rosso. In questo modo, qualsiasi dichiarazione che non passa il controllo della sintassi viene immediatamente messa in evidenza. I colori per i diversi tipi di testo vengono impostati nella finestra di dialogo mostrata nella Figura 15.1 (scegliere* Tools|Options *e selezionare la scheda* Editor Format). *A tale proposito, in VB6 viene anche visualizzato un indicatore di errore opzionale a margine della pagina se nella scheda* Editor Format *è selezionata la casella* Margin Indicator Bar.

Figura 15.1

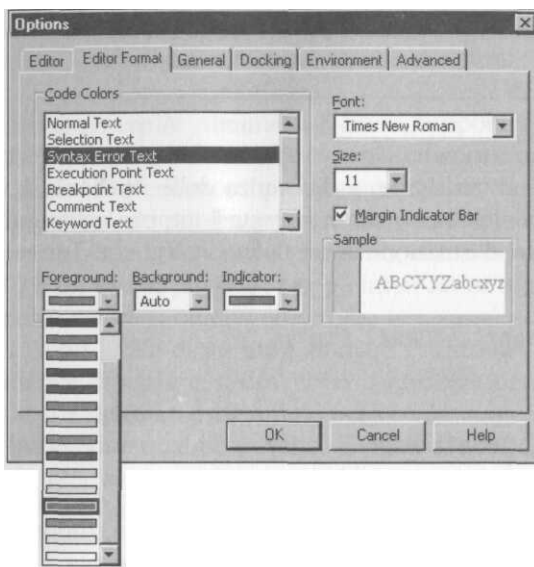
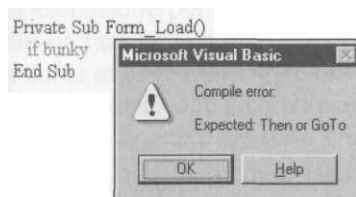
// colore con il quale sono evidenziati gli errori di sintassi viene impostato netta scheda Editor della finestra di dialogo Options.

Per esempio, se si inizia una dichiarazione If e si termina la riga senza completarla, la funzione *Auto Syntax Check* invia un messaggio di errore di compilazione che dice "Expected: Then or GoTo" (si veda la Figura 15.2).

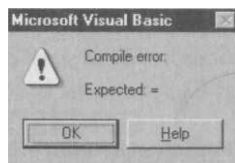
Figura 15.2

Auto Syntax Check ha rilevato un If senza Then.

Per fare un altro esempio, si supponga di creare una sottoroutine, `Bunky`, con due parametri stringa, e di cercare di chiamarla nel seguente modo:

```
Bunky ("Baby Elephant", "Small Pig")
```

Quando il cursore lascia la riga di codice che cerca di chiamare la procedura Bunky, si ottiene il messaggio di errore di sintassi mostrato nella Figura 15.3.

Figura 15.3
La chiamata a una procedura con un errore di sintassi causa un errore di sintassi.

```
Public Sub MamO
  Eimky ("Baby Elephant", "Small Pig"ì
End Sub

Public Sub Bunky(Dumbo As Stnng, Piglet As Stnng)
  Do FnmetIlffig
End Sub
```

Il messaggio di errore illustrato nella Figura 15.3 è il modo non molto chiaro in cui Visual Basic indica che le sottoroutine, a differenza delle funzioni, devono essere chiamate con la parola chiave Cali o senza parentesi intorno agli argomenti. I due modi corretti per riscrivere l'istruzione errata sono i seguenti (nessuno dei due include il segno uguale):

```
Call Bunky("Baby Elephant", "Small Pig")
```

oppure

```
Bunky "Baby Elephant", "Small Pig"
```

Come mostra l'ultimo esempio, nonostante sia evidente che, quando si riceve un messaggio di errore di sintassi, si è commesso un errore, non sempre il messaggio riflette accuratamente la natura dell'errore. Il modo migliore per individuare con esattezza il problema, se non è sufficiente riguardare il codice o leggere il messaggio di errore, consiste nel cercare l'istruzione in questione nella Guida in linea e confrontarne la definizione formale con quello che è stato scritto.
Attivando le opzioni *Require Variable Dedaration* e *Auto Syntax Check* si semplifica la gestione di questo tipo di errori.

Alcune direttive per la verifica dei programmi

I programmi di solito funzionano sulla base di intervalli di dati. Ad esempio, un programma può essere in grado di leggere il valore di un input di interi immesso dall'utente. I numeri interi in VB vanno da -32.768 a 32.767 (ovviamente, se il valore non potesse variare, non vi sarebbe motivo di leggerlo e si potrebbe utilizzare una costante).
I *limiti* di questo valore sono -32.768 e 32.767. Quando si cerca di verificare la presenza di problemi potenziali in un programma, si deve prendere in considerazione l'intervallo di tali valori. Ciò significa controllare i valori limite dell'intervallo dei

valori possibili della variabile, oltre ad alcuni valori intermedi e ad altri valori che si ritiene possano causare problemi.
Molti errori logici si verificano quando un contatore o un ciclo risulta sfalsato di uno. Si dovrebbe sospettare che vi sono errori di questo tipo (detti anche errori one-off) ogni volta che si presenta un errore logico ed è implicato un ciclo o un contatore. Un sintomo comune degli errori di sfalsamento di uno è un indice di un array (o di una casella di riepilogo) che esce dai limiti in condizioni apparentemente casuali. Un errore comune come questo è l'avvio del contatore di un ciclo da zero in un punto e da uno in un altro punto. Utilizzando delle costanti come argomenti di limite si evita questo problema. La presenza di valori non abbinati in un array, in particolare quando sono sfalsati solo di alcune posizioni, è un altro segno di questo problema.
Come esempio di un errore comune, ma grave, supponete che un programma legga un valore che rappresenta il numero di un elemento in un inventario. Dopo aver aggiunto il costo dell'elemento, il programma calcola il costo medio degli elementi eseguendo una divisione per il numero di elementi nell'inventario. Questo non è un problema se vi sono 1, 1.500 o 2.000 elementi, ma cosa succede se l'utente immette O? Anche se O è un valore possibile per gli interi senza segno, se si esegue una divisione per questo valore si ottiene un errore.
Controllare i valori limite di un programma è essenziale. In generale vi sono limiti per ogni variabile cruciale e si dovrebbe controllare ogni combinazione di questi valori quando si esegue un programma per vedere come interagiscono. Ciò è particolarmente imFortante quando si tratta di indici di matrici. Si dovrebbero verificare anche i valori intermedi, perché si potrebbe scoprire che anche qualche combinazione di questi valori genera errori inaspettati.
Più a lungo si verifica un programma in condizioni normali e anomale, più fiducia si deve avere nel programma. A mano a mano che i programmi diventano più complessi, questa fase di collaudo si allunga sempre più. Questo è il motivo per cui la principali società di software spesso forniscono migliaia di versioni preliminari dei loro programmi per il beta testing e utilizzano software per il controllo automatizzato degli errori che verificano tutti gli input possibili.
Si deve anche cercare di duplicare ogni problema di esecuzione che potrebbe verificarsi in fase di esecuzione per vedere come reagisce un programma. Questo purtroppo è difficile da fare nella realtà, considerata l'immensa varietà di sistemi e di modi diversi in cui si utilizza il software. Certamente, il tempo passato a cercare di "mandare in tilt" un'applicazione è certamente ben speso.
Nella vita reale le operazioni sui file sono una delle cause principali degli errori. Ad esempio, cosa succede se il disco è pieno e l'utente cerca di scrivere su di esso? Cosa succede se il file specificato non esiste o se è di sola lettura? E se il disco è stato rimosso o se l'utente chiede all'applicazione di scrivere in un file il record -15? Naturalmente è difficile generare tutte le circostanze problematiche plausibili, ma più ci si avvicina a questo obiettivo, più professionali saranno le applicazioni che si creano.

On Error, Resume e Resume Next

A livello procedurale, si usa l'istruzione On Error per permettere la gestione degli errori e per specificare la posizione di una routine di gestione degli errori all'interno di una procedura.

- **On Error GoTo *riga.*** Questa istruzione attiva la gestione degli errori nella procedura corrente. Quando nella procedura si verifica un errore di esecuzione, il controllo si sposta nella posizione specificata da *riga,* un'etichetta o un numero di riga (la tendenza moderna è di utilizzare etichette descrittive delle righe al posto dei numeri di riga).
- **On Error Resume Next**. L'utilizzo di questa istruzione comForta che, quando avviene un errore di esecuzione, il controllo venga trasferito all'istruzione successiva a quella che ha causato l'errore.
- **On Error GoTo ().** Questa istruzione disattiva tutti i gestori di errore attivati nella procedura corrente.

Senza la dichiarazione On Error, qualsiasi errore di esecuzione è *irreversibile.* Gli errori irreversibili fanno sì che venga visualizzato un messaggio e che termini l'esecuzione del programma.
La dichiarazione On Error Resume Next si utilizza quando si desidera che l'esecuzione del programma continui indipendentemente da tutto. Il pericolo di questo approccio è che non fornisce informazioni su potenziali problemi. Microsoft raccomanda di utilizzare On Error Resume Next quando si accede agli oggetti; tuttavia è possibile creare un gestore di errore che ripristina il funzionamento con l'enunciato successivo solo per certi tipi di errori:

```
PrivateSubGetThatRemoteThing()
   On Error GoTo ErrHandle

   Exit Sub
ErrHandle:
   If Err >= 429 And Err <= 451 Then 'Errore di automazione OLE
      ResumeNext
   Else
      Resume
   End If
End Sub
```

Le routine di gestione degli errori utilizzano le proprietà e i metodi dell'oggetto Err (descritto in maggiore dettaglio più avanti in questo capitolo). La proprietà predefinita dell'oggetto Err è Number, pertanto i riferimenti a Err nel codice precedente sono riferimenti impliciti a Err.Number (gli errori e i relativi numeri sono riFortati nell'argomento "Trappable Errors" nella Guida in linea di VB. Alcuni errori comuni con i relativi numeri sono elencati in questo capitolo nella Tabella 15.3).

È necessario assicurarsi di inserire nella procedura una istruzione Exit Sub *o* Exit Function *prima dell'etichetta digestione degli errori, in modo che l'esecuzione normale non "cada" erroneamente nel codice digestione degli errori:*

```
Public Sub Main()
   On Error GoTo ErrHandler
   Call Bunky("Baby Elephant", "Small Pig")

   Exit Sub 'Termina normalmente se non ci sono errori
ErrHandler:
   'Affronta gli errori
End Sub
```

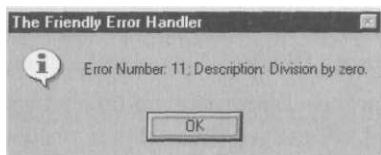
Esistono tre forme possibili per la dichiarazione Resume che, se chiamata al di fuori di un gestore di errore, causa un errore:

- **Resume.** Questa istruzione riprende l'esecuzione con l'enunciato che ha causato l'errore. Se non è stata presa alcuna misura per risolvere l'errore, si causa un errore irrversibile perché, come spiegato di seguito, un gestore di errori può elaborare un errore solo alla volta.
- **Resume Next**. Questa istruzione riprende l'esecuzione con l'enunciato successivo all'istruzione che ha causato l'errore.
- **Resume *line.*** Questa istruzione riprende l'esecuzione a un'etichetta o a un numero di riga (che deve trovarsi all'interno della stessa procedura del gestore di errore).

Un gestore di errore chiamato in causa con un'istruzione On Error è detto *abilitato.* Un gestore di errore abilitato che viene chiamato a seguito di un errore è detto *attivo.* Un gestore di errore è in grado di gestire un solo errore alla volta. Supponete che si verifichi un errore mentre è attivo un gestore di evento (supponiamo che sia l'errore numero due). Il gestore di errori corrente non può gestire il secondo errore. L'esecuzione scorre a ritroso la catena di procedure del programma finché trova un gestore di errore abilitato ma inattivo che gestisca l'errore numero due. La procedura che contiene il gestore abilitato ma inattivo diventa la procedura attiva. Se questo gestore non viene trovato, l'errore diventa irreversibile.
Per sviluppare e verificare programmi, di solito è sufficiente il codice di gestione degli errori che utilizza le proprietà **Name** e Description dell'oggetto Err per visualizzare un messaggio e che riprende l'esecuzione all'istruzione successiva. Questo tipo di gestione degli errori informa lo sviluppatore che potrebbe esistere un problema potenziale, permettendo comunque al programma di eseguire la maggior parte di codice possibile. La gestione degli errori progettata per un programma nella release generale dovrebbe essere più sofisticata e se possibile permettere agli utenti di risolvere gli errori. Per esempio, se il drive A non contiene un disco, l'utente dovrebbe essere in grado di inserirne uno e riprendere l'esecuzione del programma.

Nella seguente routine, se l'utente immette zero in Text1 *(o se lascia il controllo vuoto), viene generato un errore di divisione per zero (si noti che le routine di gestione degli errori in questo capitolo sono parte di un progetto salvato nel CD-ROM con il nome Error. Vbp). Il gestore di errore passa un messaggio appropriato (si veda la Figura 15.4) e prosegue l'esecuzione con l'enunciato successivo a quello che ha generato l'errore* (MsgBox "I'm Next!"*):*

```
Private Sub Command1_Click()
   On Error GoTo ErrHandle
   Dim x, y, z
   x = Val(Text1)
   z = 1 : y = 1
   z = y / x
   MsgBox "I'm Next!"
   Exit Sub
ErrHandle:
   MsgBox "Error Number:" + Str(Err.Number) + _
      "; Description: " + Err.Description + _
         ". ", vbInformation, _
      "The Friendly Error Handler"
   Resume Next
End Sub
```

Figura 15.4 *Un messaggio abbastanza esplicativo generato da un gestore di errore.*

Se la proprietà Number di Err è zero (implicitamente, Err = 0), significa che nella procedura corrente non è stato generato alcun errore. La generazione di un errore è un evento, pertanto è possibile pensare a un errore che viene generato. Come discusso più avanti, *generare* un errore significa provocare quell'errore, esattamente come generare un evento con la dichiarazione RaiseEvent significa provare l'evento stesso.
È possibile scoprire se vi sono errori nella procedura corrente verificando se la proposizione Err = 0 è vera o falsa, come mostrato nel Listato 15.1.

Listato 15.1 *Verifica di una condizione di errore.*

```
Private Sub Command2_Click()
   On Error Resume Next
   Dim x, y, z
   Dim GoodNews As Boolean
   x = Val(Text1)
   z = 1 : y = 1
   z = y / x
   GoodNews = (Err = 0)  'Assegna valori di posizione a GoodNews
   If Not GoodNews Then
      MsgBox "Qualcosa è andato storto: forse divisione per zero!"
   Else
      MsgBox "Va tutto bene!"
   End If
End Sub
```

L'oggetto Err

L'oggetto Err contiene informazioni sugli errori di esecuzione e ha le proprietà e metodi elencati nelle Tabelle 15.1 e 15.2.

Tabella 15.1 *Proprietà dell oggetto Err.*

Proprietà	Scopo
Number	Restituisce o imposta il numero dell'errore. La gamma di valori validi è 0-65.535.
Source	Imposta o restituisce il nome dell'oggetto o dell'applicazione che ha generato l'errore, ad esempio il nome del progetto corrente di VB.
Description	Contiene una descrizione dell'errore sulla base del numero di errore. Se la stringa non esiste, la proprietà Description contiene la stringa "Application-defined or object-defined error."
HelpFile	File della Guida in linea del progetto.
HelpContext	ID di contesto del file della Guida in linea.
LastDLLError	Restituisce un codice di errore di sistema prodotto da una chiamata a una DLL che ha restituito un flag di errore (solo a 32 bit).

Tabella 15.2 *Metodi dell'oggetto Err.*

Metodo	Scopo
Clear	Cancella tutte le proprietà dell'oggetto Err (questo metodo equivale alla dichiarazione Err = 0 nelle precedenti versioni di Visual Basic). Si dovrebbe utilizzare il metodo Err.Clear per cancellare esplicitamente l'oggetto Err dopo aver gestito un errore. VB chiama automaticamente questo metodo ogni volta che viene eseguita una funzione Resume, Exit Sub, Exit Function, Exit Property o On Error.
Raise	Genera un errore di esecuzione. Può essere utilizzato anche per impostare le proprietà dell'oggetto Err, come descritto nella Tabella 15.1.

La funzione Error

La sintassi utilizzata nelle Versioni 1, 2 e 3 di Visual Basic, nelle quali la funzione Err restituiva il numero di errore corrente e Error(Err) restituiva la descrizione dell'errore, è stata mantenuta per permettere la compatibilita con tali versioni. Poiché Number è la proprietà predefinita dell'oggetto Err corrente, Err stesso restituisce il numero di errore. Ciò significa che la funzione Error restituisce ancora una descrizione dell'errore, quando viene chiamata utilizzando Err come argomento.

Il metodo Raise

Il metodo Raise utilizza i seguenti argomenti: *Numero, Sorgente, Descrizione, FileDiGuida* e *TestoDiGuida.* Tutti, ad eccezione di *Number,* sono opzionali. *Numero* è un intero lungo che identifica l'errore. Gli errori di Visual Basic interni e quelli definiti dall'utente vanno da 1 a 65.535 (gli errori definiti dall'utente sono discussi in maggior dettaglio più avanti).

Errori intercettabili comuni

Nella Tabella 15.3 sono elencati alcuni errori comuni intercettabili in Visual Basic. Si tratta di un elenco abbreviato ed *è* possibile trovare ulteriori errori ricercando le categorie sotto "Trappable Errors" nella Guida in linea di VB. Inoltre anche i controlli personalizzati e altri oggetti possono avere i loro valori di errore.

Tabella 15.3 *Alcuni errori intercettabili.*

Numero di errore	Descrizione
3	Return without GoSub
5	Invalici procedure Call
6	Overflow
7	Out of memory
9	Subscript out of range
10	Array is fixed or temForarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
16	Expression too complex
17	Cannot perform requested operation
20	Resume without error
28	Out of stack space
51	Internai error
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
59	Bad record length
61	Disk full
62	Input past end of file
63	Bad record number
67	Too many files
68	Device unavailable
70	Permission denied
71	Disk not ready

Numero di errore	Descrizione
74	Cannot rename with different drive
75	Path/file access error
76	Path not found
91	Object variable or With block variable not set
298	System DLL [*dll name]* could not be loaded
321	Invalid file format
335	Could not access system registry
336	Object server not correctly registered
3,37	Object server not found
340	Control array element does not exist
341	Invalid control array index
2342	Not enough room to allocate control array item
343	Object is not an array
344	Must specify index when using object array
360	Object is already loaded
361	Can't load or unload this object
362	Controls created at design time cannot be unloaded
380	Illegal property value
381	Illegal property array index
384	A form cannot be moved when minimized or maximized
401	Can't show nonmodal form when modal form is displayed
419	Permission to use object denied
421	Method not applicable for this object
423	Property or method not found
424	Object required
426	Only one MDI form allowed
427	Invalid object type; Menu control required
428	Pop-up menu must have at least one submenu
429	OLE automation server cannot create object
430	Class does not support OLE automation
432	Filename or class name not found during OLE automation operation
438	Object doesn't support this property or method
440	OLE automation error
445	OLE does not support this action
449	Argument not optional
450	Wrong number of arguments
451	Object not a collection
24574	No fonts exist (common dialog error)
28664	No printer device drivers were found (common dialog error)

(continua)

Tabella 15.3 *Alcuni errori intercettabili.* *(continua)*

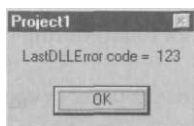
Numero di errore	Descrizione
28670	Load of required resources failed (common dialog error)
31001	Out of memory
31004	No object
31027	Unable to adivate object
31032	Unable to create embedded object
31036	Error saving to file
31037	Error loading from file

La proprietà LastDLLError

La proprietà LastDLLError, di sola lettura, restituisce il codice di un errore di sistema prodotto dalla chiamata a una libreria a collegamento dinamico (DLL). Quando restituisce un flag Failure che può essere recuperato, la funzione della DLL imposta un codice di errore. Ad esempio, il seguente codice chiama la funzione GetDiskFreeSpace, discussa nel Capitolo 11:

```
Private Sub Form_Click()
Dim RetVal As Boolean
    RetVal = GetDiskFreeSpace("Hacker", 2, 3, 4, 5)
    If RetVal Then
        MsgBox "Function Call Succeeded"
    Else
        MsgBox "LastDLLError code = " + Str(Err.LastDllError)
    End If
End Sub
```

Ovviamente, questa chiamata non giunge a buon fine, perché la funzione si aspetta come primo argomento un'espressione che ha come valore un drive radice (si veda la Figura 15.5). Per determinare cosa significa il codice di errore, bisogna fare riferimento alla documentazione della DLL.

Figura 15.5 *È fallita la chiamata a una DLL.*

Generazione di errori

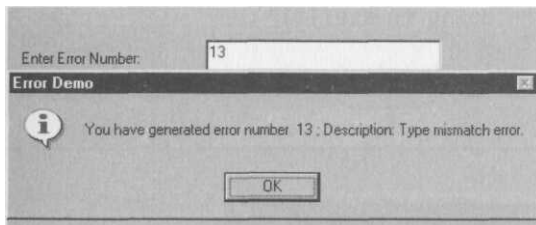
Per verificare le routine di gestione degli errori è necessario essere in grado di creare errori. Alcuni di essi, ad esempio la divisione per zero discussa precedentemente in questo capitolo, sono semplici da creare. Altri, ad esempio l'errore 28 "Out of stack space" o l'errore 61 "Disk full", possono essere complicati da creare (o addirittura impossibili).

Fortunatamente, con il metodo Raise dell'oggetto Err è semplice simulare qualsiasi errore (in realtà "simulare" non è la parola corretta: dal punto di vista del programma, quando si genera un errore, l'errore si verifica effettivamente). Come mostrato nel Listato 15.2, generare un errore è semplice.

Listato 15.2 *Generare un errore.*

```
Private Sub cmdRaise_Click()
   On Error GoTo ErrMessage
   Err.Raise Val(Text1)
   Exit Sub
ErrMessage:
   MsgBox "You have generateci error number " & _
      Str(Err.Number) & " ; Description: " & Err.Description & _
      " error.", vbInformation, "Error Demo"
   ResumeNext
End Sub
```

Se si immette un numero, ad esempio 13, e si esegue la procedura, viene creati ("generato") l'errore immesso, come mostrato nella Figura 15.6.

Figura 15.6 *Utilizzando i metodi dell'oggetto Err è semplice "generare" un errore.*

Se si chiama il metodo Raise *dell'oggetto Err con l'argomento* 0, Err.Number *non equivale a* 0. *Poiché* Err.Number = 0 *significa che non sono stati generati errori passando 0 comeparametro del metodo* Raise *si causa un errore.* Err.Raise 0 *fa si che* Err.Number *sia uguale a 5 ("Invalidprocedure call").*

Generazione errori definiti dall'utente

Gli errori definiti dall'utente sono errori personalizzati per i quali si deve inventare il numero e la descrizione; si utilizzano quando si devono intercettare errori non integrati in VB e nell'oggetto Err. In altre parole, il codice stesso genera l'errore in determinate condizioni. L'utilizzo degli errori personalizzati è importante in tutti i programmi complessi; quando si crea un'applicazione o un controllo ActiveX, è particolarmente importante creare errori che forniscono informazioni su ciò che 1i ha causati.

Il metodo Err.Raise viene utilizzato per assegnare il numero e la descrizione agli errori definiti dall'utente. In teoria si può utilizzare qualsiasi numero libero tra 1 e 65.535 inclusi (l'intervallo dei numeri validi della proprietà Err.Number). Per esem-

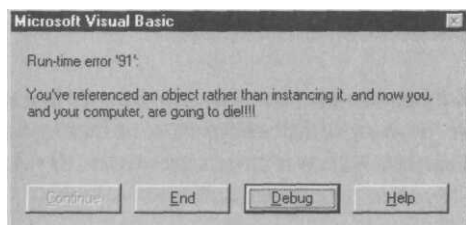
pio, non esistono i numeri di errore predefiniti 1, 2 e da 21 a 27. Per evitare qualsiasi conflitto con i numeri di errore predefiniti, è consigliabile tuttavia iniziare a numerare gli errori definiti dall'utente dal numero più alto, 65.535, e quindi scendere:

```
Const MyBigBadError = 65535
Const MyNextError = MyBigBadError - 1
```

Si può provare a utilizzare il metodo Raise *per modificare la descrizione di un numero di errore esistente (si veda il messaggio nella Figura 15.7). Come mostrato nel Listato 15.3, questo codice cambia la descrizione dell'errore 91 ("Object variable or With block variable not set") e quindi genera l'errore.*

Listato 15.3 *Modifica della descrizione di un errore predefinito.*

```
Private Sub Form_Click()
   Dim Silly As String
   Debug.Print txtDesc ' Mostra il valore di txtDesc.text nel
                        ' pannello inferiore della finestra di debug
   Silly - "You've referenced an object"
   Silly = Silly + "rather than instancing it,"
   Silly = Silly + " and now you, and your computer,"
   Silly = Silly + " are going to die!!!!"
   Err.Raise 91, , Silly
   Dim x As Form1
   x.Caption = "blah"
End Sub
```

Figura 15.7 *È possibile modificare la descrizione degli errori.*

La modifica del testo dei messaggi di errore può risultare utile per rendere il testo più descrittivo di quello originale.

Il concetto originale è che la proprietà Number *di un errore definito dall'utente in un oggetto di automazione OLE dovrebbe partire da* vbObjectError *(-2147221504). Visual Basic utilizza i numeri di errore fino a* vbObjectError + *512. Si dovrebbe iniziare a numerare gli errori personalizzati per un server di automazione OLE (ad esempio applicazioni e controlli ActiveX) a partire da:*

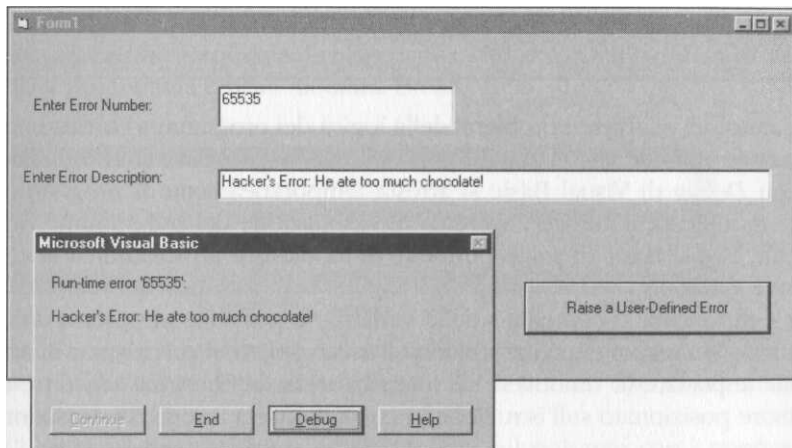
```
Const MyFirstError = 1 + vbObjectError + 512
Const MyNextError = MyFirstError + 1
```

La procedura mostrata nel Listato 15.4 genera un errore sulla base di un numero e di una descrizione immessi dall'utente (Textl e txtDesc sono caselle di testo) e quindi visualizza l'errore.

Listato 15.4 *Generare un errore definito dall'utente.*

```
Private Sub cmdRUDE_Click()
    DimSAsString
    Err.Raise Textl, , txtDesc
    S = "The following Error occurred:" + vbCrLf + vbCrLf
    'add the error string
    S = S + Err.Description + vbCrLf
    'add the error number
    S = S + "Number: " + CStr(Err.Number)
    'bip e indica l'errore
    Beep
    MsgBox S, vbExclamation, "User-Defined Error Demonstration"
    ResumeNext
End Sub
```

Le possibilità di creazione di errori personalizzati sono illimitate (Figura 15.8)!

Figura 15.8 *È possibile utilizzare il metodo Raise dell'oggetto Err per definire e memorizzare un errore personalizzato, in questo caso l'errore numero 65535: "Hacker's Error: He ate too much chocolate!"*

Strumenti di debugging

Gli errori logici in un programma di norma sono più difficili da trovare rispetto agli errori di compilazione o di esecuzione. Un errore logico può trovarsi molto all'interno di una lunga catena di chiamate a procedure complesse, di rapForti tra gli oggetti e di dichiarazioni. Dietro agli errori logici più ostinati spesso vi sono quelle assunzioni a cui si fa molta fatica a rinunciare, le classiche *idee fisse.* Una mente flessibile e la volontà di capire le cause e gli effetti un passo alla volta sono i prerequisiti per poter terminare positivamente il debugging. Fortunatamente in Visual Basic sono inclusi alcuni strumenti che aiutano a individuare gli errori nella logica.

Le caselle di messaggio e Debug.Print

La funzione MsgBox può essere utilizzata per visualizzare valori variabili durante l'esecuzione di un programma. È molto semplice inserire in punti strategici delle caselle di messaggio temForanee che visualizzano il contenuto di qualsiasi variabile di cui non si è sicuri. In questo modo è possibile utilizzare una semplice casella di messaggio come un'imFortante strumento di debugging.
Il metodo Print dell'oggetto Debug viene utilizzato in modo molto simile. Le espressioni che fanno parte degli argomenti di Debug.Print vengono visualizzate nel pannello *Immediate* mentre vengono eseguite le istruzioni Debug. I vantaggi dell'utilizzo di Debug.Print sono due: non viene interrotta l'esecuzione del programma (come con le caselle di messaggio) ed è possibile lasciare inserite le istruzioni Debug, senza compromettere le prestazioni, durante la compilazione.

Accesso veloce ai comandi di debugging

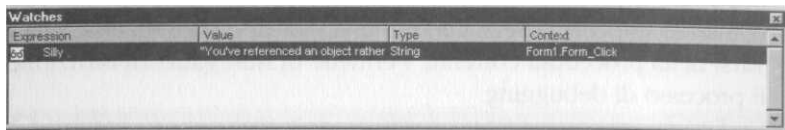
È possibile utilizzare il menu *Debug* per accedere agli strumenti di debugging, inclusi *Toggle Breakpoint, Instant Watch, Calls, Step In* e *Step Over.* Inoltre è possibile visualizzare la barra degli strumenti *Debug,* che include i pulsanti per accedere alla maggior parte delle funzioni di debugging; per visualizzarla, scegliere *Toolbars* dal menu *View* e assicurarsi che sia selezionato *Debug Toolbar.*

Un aiuto nel risolvere i problemi della logica del programma consiste nel fatto che è possibile modificare il codice senza arrestare il programma. Scegliendo *Break* dal menu *Debug* di Visual Basic si arresta temForaneamente il programma, quindi si può modificare il codice e riprendere l'esecuzione del programma. Nella modalità Break, Visual Basic di solito permette di modificare il programma fino a dichiarare nuove variabili.
Per determinare il contenuto delle variabili, è possibile impostare un'interruzione. Quando si raggiunge un'interruzione, l'esecuzione del programma si arresta. È possibile impostare (o rimuovere) le interruzioni nella finestra *Code* in tre modi. Con il cursore posizionato sull'istruzione in cui si desidera inserire l'interruzione:

- premere F9
- scegliere *Debug |Toggle|Breakpoint*
- scegliere *Toggle\Breakpoint* dal menu di scelta rapida nella finestra *Code*

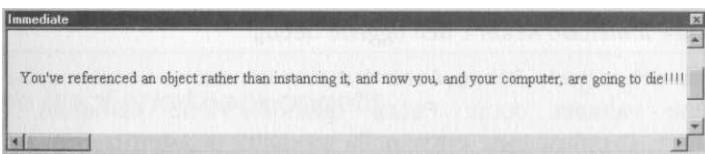
L'istruzione selezionata viene formattata in grassetto per indicare che è stata impostata un'interruzione (si noti che in realtà l'esecuzione si arresta alla fine della riga che precede l'interruzione).
Per visualizzare il contenuto delle variabili al momento dell'interruzione, impostare una *espressione di controllo (watch),* scegliendo *Add Watch* dal menu *Debug.* Quando viene raggiunta l'interruzione, il contenuto delle variabili incluse nell'elenco *Watch List* viene visualizzato nella finestra *Debug.* Questa permette di controllare i

valori di un programma durante la *modalità interruzione,* cioè quando si sceglie *Break* dal menu *Run* o quando il programma raggiunge un'interruzione.
La *finestra Watch,* chiamata anche *pannello Watch,* mostra il valore delle espressioni aggiunte all'elenco *Watch List.* Ad esempio, nella Figura 15.9 il pannello *Watch* mostra il valore della variabile *Silly* che è stata aggiunta all'elenco *Watch List.* La *finestra Immediate,* chiamata anche *pannello Immediate,* viene utilizzata per controllare le dichiarazioni di debugging aggiunte al codice utilizzando il metodo Debug.Print (si veda il riquadro "Le caselle di messaggio e Debug.Print" in questo paragrafo).

Figura 15.9
Visualizzazione del contenuto di una variabile.

Ad esempio, la seguente istruzione Debug:

```
Debug.Print Silly
```

fa sì che nel pannello *Immediate* venga visualizzato il contenuto della variabile Silly, come mostrato nella Figura 15.10 (si veda il paragrafo "Generazione di errori definiti dall'utente" precedentemente in questo capitolo). È anche possibile utilizzare il pannello *Immediate* per immettere del codice che venga eseguito immediatamente quando il programma è nella modalità Break.

Figura 15.10
Nel pannetto Immediate, visualizza l'output generato dal metodo Print.

La finestra *Locals* visualizza automaticamente tutte le variabili dichiarate nella procedura corrente e i loro valori. È possibile visualizzare la finestra *Locals* selezionandola nel menu *View* di VB.
La finestra *Calls,* che viene visualizzata facendo clic sul pulsante con i puntini di sospensione sulla destra della casella di riepilogo delle procedure nella finestra *Locals,* mostra un elenco di tutte le procedure attivate ma non terminate nel progetto. In altre parole si tratta di un elenco di chiamate a procedure nidificate che indicano come si è arrivati all'interruzione.
Altri strumenti di debugging di VB sono l'esecuzione del programma passo per passo (stepping) e la possibilità di modificare i valori delle variabili durante l'esecuzione del programma. Lo stepping singolo è uno strumento di debugging potente che permette di avere un'immagine precisa di ciò che sta facendo il programma. Nella modalità Break di VB6 vi sono numerosi altri comandi di debugging, tutti inclusi nel menu *Debug.*

- **Step Over [Esegui istruzione/routine,** Maiusc+F8]. Questo comando è simile allo stepping, ad eccezione del fatto che, quando la dichiarazione corrente contiene una chiamata a una procedura, Step Over esegue la procedura chiamata come un'unità e quindi ritorna all'enunciato successivo nel modulo corrente.
- **Run To Cursor [Esegui fino al cursore,** Ctrl+F8]. Questo comando seleziona una dichiarazione successiva nel codice dove si desidera che termini l'esecuzione. Aiuta a saltare grandi blocchi di codice che non si desidera visualizzare.
- **Set Next Statement [Imposta istruzione successiva,** Ctrl+F9]. Questo comando imposta l'esecuzione di una riga di codice diversa, che deve trovarsi nella procedura corrente. Permette di rieseguire un'istruzione durante il processo di debugging.
- **Show Next Statement [Mostra istruzione successiva].** Questo cornando posiziona il cursore nella riga che segue nell'ordine di esecuzione.

Utilizzo delle asserzioni

Un'asserzione è una dichiarazione all'interno di un programma che permette al programma di controllare il proprio codice durante l'esecuzione. Le asserzioni deve essere utilizzate come secondo controllo delle assunzioni sottostanti relative alla logica del programma.

Per creare asserzioni che non appaiono mai in un'applicazione compilata è possibile utilizzare il metodo Assert *dell'oggetto* Debug.

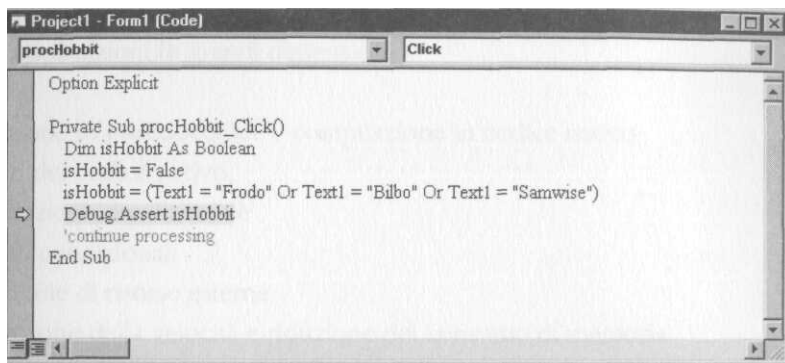
L'argomento del metodo Debug.Assert è una istruzione booleana. Se la dichiarazione viene valutata come False quando viene elaborata l'espressione Debug.Assert, l'applicazione entra nella modalità di interruzione con l'istruzione evidenziata. Ad esempio, supponete che in un programma vi sia una routine che assume che l'utente abbia immesso un nome specifico da un elenco. Si potrebbe verificare tale supposizione con un'asserzione:

Il debugging e l'input tramite il mouse e la tastiera

Prima di terminare la discussione sul debugging, si dovrebbe essere a conoscenza di un problema. Poiché i programmi di VB sono "guidati dagli eventi", se si inserisce un'interruzione in una procedura di evento MouseDown o KeyPress e si rilascia il pulsante del mouse o il tasto mentre il programma è nello stato Break, potrebbe succedere che quando il codice riprende non si verifichi mai l'evento MouseUp o Keyllp. In altre parole, è necessario tenere a mente che i programmi di Visual Basic rispondono all'ambiente di Windows e se si modifica questo ambiente durante il debugging, si potrebbero ottenere risultati inaspettati.

```
Private Sub procHobbit_Click()
   Dim isHobbit As Boolean
   isHobbit = False
   isHobbit = (Text1 = "Frodo" Or Textl = "Bilbo" Or _
      Textl = "Samwise")
   Debug.Assert isHobbit
   'continua l'elaborazione
End Sub
```

Quando il programma viene eseguito nell'ambiente di VB, l'elaborazione continua senza interruzione, a meno che sia stato immesso un nome non valido. In questo caso, quando VB valuta la dichiarazione Debug.Assert, scopre che l'argomento è False, entra nella modalità di interruzione ed evidenzia l'asserzione, come mostrato nella Figura 15.11.

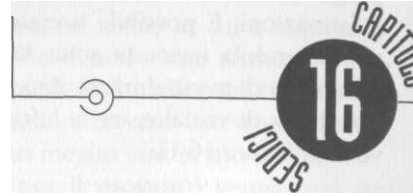
Figura 15.11 *Si dovrebbe utilizzare il metodo DebugAssert in modo che i programmi verifichino le assunzioni logiche sottostanti nel codice durante*

Gestione degli errori negli oggetti

Quando le applicazioni utilizzano numerosi oggetti, per esempio controlli ActiveX, può essere difficile determinare la natura di un errore e quale oggetto lo genera. Di conseguenza è imFortante includere una gestione appropriata degli errori nei server e nei controlli ActiveX, in particolare se gli oggetti devono essere distribuiti ad altri sviluppatori.

Riepilogo

In questo capitolo si è discusso del debugging e degli strumenti di debugging disponibili in Visual Basic. Si è iniziato parlando dei tre tipi di errori in cui ci si può imbattere: gli errori di sintassi e di compilazione, gli errori di esecuzione e gli errori logici (di progettazione). Quindi si è parlato della gestione degli errori di esecuzione, delle tecniche di debugging e di come si possono utilizzare in modo efficiente gli strumenti forniti da Visual Basic.

- Si è appreso come utilizzare le istruzioni On Error GoTo, On Error Resume Next e On Error GoTo 0 nelle procedure.
- Si è discusso delle istruzioni Resume e Resume Next.
- Si è discusso della numerazione degli errori.
- Sono stati spiegati l'oggetto Err e le sue proprietà e i suoi metodi.
- Si è appreso come utilizzare il metodo LastDLLError dell'oggetto Err.
- Si è discusso di come si generano gli errori.
- Si è discusso di come si generano gli errori definiti dall'utente.
- Si è appreso come attivare e le interruzioni e come utilizzare le finestre *Immediate, Watch* e *Local.*
- Si è appreso come utilizzare gli strumenti per l'esecuzione passo per passo.
- Si è parlato dell'utilizzo delle asserzioni.

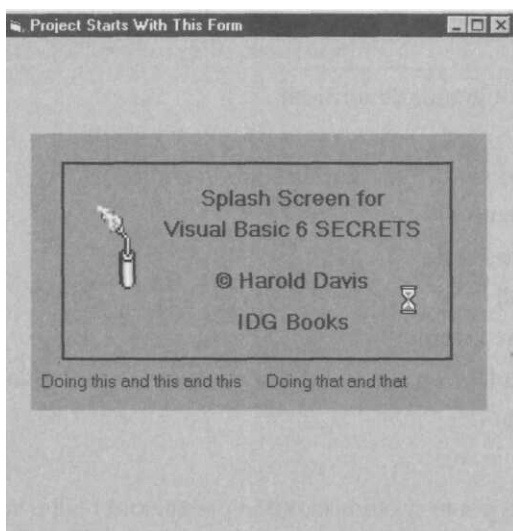
OTTIMIZZAZIONE DEI PROGRAMMI

- Le schermate di avvio
- Avvio di applicazioni di grandi dimensioni
- Lo shelling
- Compilazione in pseudocodice e compilazione in codice nativo
- Gli switch del codice nativo
- La compilazione condizionale
- Le costanti condizionali
- Utilizzo di file di risorse esterne
- Ottimizzazione della velocità e riduzione del consumo di memoria
- Ricerca di file sul disco
- La ricorsione

In questo capitolo vedremo come si ottimizzano le prestazioni reali e apparenti dei programmi. Per "prestazioni apparenti" si intende la velocità dell'applicazione percepita dall'utente. Spesso questa è più imFortante della velocità reale dell'esecuzione del codice. Se si mette a conoscenza l'utente di ciò che avviene, ad esempio che è in corso un caricamento, un'inizializzazione, una convalida o una connessione, è più probabile che l'utente sia maggiormente disposto a perdonare dei ritardi. Vengono anche discusse le conseguenze della compilazione del codice nativo e il significato degli switch *Advanced Optimizations* della compilazione del codice nativo.

Quando si distribuisce un'applicazione, è ragionevole che l'utente assuma che questa si comForta in modo civile, vale a dire che i programmi non devono rovinare dischi fissi, diffondere virus o far perdere tempo inutilmente all'utente. Inoltre 2 imFortante la comunicazione. Il rapForto tra l'applicazione e l'utente, come tutti i rapForti, funziona molto meglio se l'applicazione comunica (per esempio visualizzando il messaggio "Caricamento in corso" mentre sono in corso operazioni che richiedono molto tempo).

Le schermate di avvio

Le schermate di avvio *(splash screen)* sono utilizzate per tenere informato o per intrattenere l'utente durante il caricamento iniziale del programma. Mentre è visualizzata una schermata di avvio, può essere effettuato il caricamento dei form di un progetto, l'accesso a database, l'inizializzazione di database locali o la convalida di informazioni. È possibile eseguire qualsiasi operazione che richiede molto tempo mantenendola nascosta sotto la schermata di avvio. Inoltre, se si visualizza una schermata di avvio durante il caricamento di un'applicazione professionale, si ha la possibilità di visualizzare le informazioni sull'autore, sull'editore e sul copyright (si veda la Figura 16.1).

Figura 16.1 *È semplice creare schermate di avvio di bell'aspetto che rendono il codice inizializzazione più piacevole per gli utenti.*

Le schermate di avvio di solito sono form senza bordi con etichette per visualizzare informazioni e controlli per visualizzare la grafica. È possibile utilizzare controlli timer per includere animazioni o per aggiornare le informazioni visualizzate.
Il modo più semplice per caricare una schermata di avvio da `Sub Main` è di chiamare il metodo `Show` della schermata di avvio in modo modale, per poi scaricarlo utilizzando un controllo timer. La proprietà `Interval` del controllo timer deve essere impostata a un valore ragionevole, per esempio 5000 (cinque secondi) e ci si deve assicurare che la sua proprietà `Enabled` sia impostata su `True`. Per esempio:

```
Sub Main ()
   frmSplash.Show vbModal
   'carica i form del programma dopo che frmSplash è stato scaricato
   frmStartProject.Show
End Sub

Sub tmrSplash_Timer()
   'L'intervallo è finito - scarica frmSplash
```

```
    Unload Me
End Sub
```

Impostando una schermata di avvio in questo modo, tuttavia, diventa più diffìcile implementare il vero motivo per cui si aggiunge una schermata di avvio: l'esecuzione di codice mentre è visualizzata la schermata di avvio. Idealmente (poiché è il modo più semplice) il punto di inserimento di questo codice dovrebbe essere l'evento Load di frmStartProject. Tuttavia, a causa del modo in cui è impostato l'esempio, frmStartProject non viene caricato finché non viene scaricato frm-Splash. Si potrebbero inserire delle chiamate per avviare il codice da frmSplash, ma si tratta una soluzione un po' pasticciata. Nel migliore dei casi l'utente dovrebbe aspettare il caricamento dei forni successivi. È molto meglio fare in modo che frm-StartProject (e qualsiasi altro forni necessario per il progetto) venga caricato mentre è visualizzato frmSplash.

*// modo in cui ho impostato ilprogetto di esempio, salvato nel CD-ROM con il nome Splash. Vbp, consiste nel visualizzare f*rmSplash *comeform non modale. L'inizializzazione e il caricamento delform* frmStartProj ect *continuano immediatamente.*

Poi ho dovuto utilizzare la funzione SetWindowPos dell'API (si veda il Capitolo 11) per assicurarmi che frmSplash rimanesse in primo piano rispetto agli altri form che venivano caricati. Inoltre in Sub Main ho impostato la proprietà Enabled di frm-StartProject su False, in modo che l'utente non potesse accedere agli eventi e ai controlli del form finché non fosse completata l'inizializzazione (anche gli altri form sotto la schermata di avvio devono essere impostati in questo modo).
Nell'esempio viene inoltre impostato il cursore con la clessidra (Screen.Mouse-Pointer); il cursore torna alla forma predefinita solo quando la schermata di avvio viene scaricata. Per tenere aggiornata la schermata di avvio sono stati aggiunti dei timer secondari (tmrThis, tmrThat), che espandono le etichette lblThis e lbl-That. Nello stesso modo è possibile animare il controllo immagine. La dichiarazione dell'API e il codice Sub Main sono:

```
Option Explicit
 ' Costanti per i parametri dell'API SetWindowPos
Public Const HWND_TOPMOST = -1
Public Const SWP_NOSIZE = &H1
Declare Function SetWindowPos Lib "User32" _
    (ByVal hWnd As Long, ByVal hWndInsertAfter As Long, _
   ByVal x As Long, ByVal Y As Long, _
   ByVal ex As Long, ByVal cy As Long, _
   ByVal wFlags As Long) As Long

Sub Main()
    'Il progetto viene impostato a StartUp da Sub Main()
   frmStartProject.Enabled = False
   Screen.MousePointer = vbHourglass
   frmSplash.Show
     Mostra frmSplash durante il caricamento del form principale
   frmStartProject.Show
End Sub
```

Il Listato 16.1 contiene il codice in frmSplash.

Listato 16.1 *Creazione del codice per una schermata di avvio.*

```
Option Explicit
Private Sub Form_Load()
   Dim lTop As Long, lLeft As Long, RetVal as Long
   'Centra la schermata di avvio e stabilisce la posizione in pixel
   lTop = (Screen.Height \ 2 - Me.Height \ 2) \ _
      Screen.TwipsPerPixelY
   lLeft = (Screen.Width \ 2 - Me.Width \ 2) \ _
      Screen.TwipsPerPixelX
   RetVal = SetWindowPos(Me.hWnd, HWND_TOPMOST, lLeft, _
      lTop, 0&, 0&, SWP_NOSIZE)
End Sub

Private Sub tmrSplash_Timer()
   ' Scarica questo form quando l'intervallo è finito.
   'Abilita il forni del progetto
   frmStartProject.Enabled = True
   'Ripristina il cursore
   Screen.MousePointer = vbDefault
   'Scarica la schermata di avvio
   Unload Me
End Sub

Private Sub tmrThat_Timer()
   'Aggiorna lblThat
   lblThat = lblThat + "and that "
End Sub

Private Sub tmrThis_Timer()
   'Update lblThis
   lblThis = lblThis + "and this "
End Sub
```

I metodi TwipsPerPixel

TwipsPerPixelX restituisce il numero di twip per pixel di un oggetto misurato orizzontalmente e TwipsPerPixelY restituisce i twip per pixel di un oggetto misurato verticalmente. È pratica comune utilizzare questi metodi per convertire le misure in pixel, come richiesto dalla maggior parte delle funzioni di visualizzazione dell'API.

Avvio di un'applicazione di grandi dimensioni

È possibile utilizzare una schermata di avvio per caricare un eseguibile completamente separato mediante l'istruzione Shell. Il trucco consiste nel dividere in due il programma: un piccolo eseguibile che carica le librerie di esecuzione di VB e la schermata di avvio e un eseguibile di dimensioni maggiori dall'altra. Così si riduce il tempo totale apparente di caricamento, cioè la quantità di tempo che sembra necessario perché diventi attivo il forni dell'applicazione.

È possibile implementare lo shelling in un timer con la proprietà .Interval *impostata in modo appropriato. Ad esempio, il seguente codice, incluso in ShellSpl.Vbp nel CD-ROM allegato al libro, carica Notepad dopo che il* Form1 *di VB è rimasto sullo schermo per due secondi:*

```
Private Sub tmrShell_Timer()
    Dim Retval As Long, FileName As String
    FileName = "Notepad.Exe"
    Retval = Shell(FileName, vbNormalFocus)
    Unload Me
End Sub
```

Sostituendo Notepad.Exe con un altro eseguibile di VB di grandi dimensioni si ha un programma di lancio in cui ogni parte impiega circa la metà del tempo che normalmente sarebbe necessario solo per caricare il programma. Potendo utilizzare la prima parte per visualizzare un messaggio ("Si prega di attendere: caricamento in corso"), si tratta di una tecnica che vale la pena di prendere in considerazione.

La funzione Shell

Probabilmente avete già familiarità con la funzione Shell (si veda l'esempio precedente), utilizzata per eseguire un programma esterno. Il problema principale quando si utilizza Shell in un'applicazione, se si desidera tornare all'applicazione avviata con la funzione Shell, consiste nel mantenere il controllo del flusso di esecuzione dopo averlo ceduto a un'applicazione esterna. Quando, precisamente, il controllo dell'esecuzione torna al codice che ha eseguito la funzione?
Di norma si fa tornare l'esecuzione al programma che ha eseguito la funzione Shell quando viene chiuso il programma esterno. In Win32 però la cosa funziona diversamente rispetto ai sistemi operativi a 16 bit, come vedremo tra poco.

La funzione Shell *restituisce dati del tipo intero lungo. Nonostante in alcune parti della documentazione sia indicato diversamente,* Shell *non apre applicazioni in base a file di documento associati (per esempio non è possibile passare a* Shell *un file .Doc per lanciare Word)).*

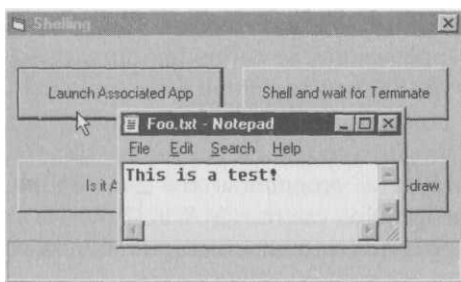
I programmi che dimostrano la funzione Shell *sono salvati nel CD-ROM allegato al libro con il nome ShellTer.Vbp.*

Lancio di un'applicazione mediante una associazione di file

Per lanciare un'applicazione utilizzando un file associato è possibile utilizzare la funzione ShellExecute dell'API. Ad esempio, per lanciare Notepad con un file chiamato Foo.Txt caricato, ammesso che Foo.Txt si trovi nella posizione specificata (in questo esempio la directory radice C:\), si può utilizzare il codice nel Listato 16.2 (si veda la Figura 16.2):

Listato 16.2 *Lancio di un'applicazione utilizzando un file associato.*

```
Option Explicit
'Dichiara ShellExecute e le costanti
Private Declare Function ShellExecute Lib _
    "shell32.dll" Alias "ShellExecuteA" _
    (ByVal hwnd As Long, _
    ByVal lpOperation As String, _
    ByVal lpFile As String, _
    ByVal lpParameters As String, _
    ByVal lpDirectory As String, _
    ByVal nShowCmd As Long) As Long
Private Const SW_SHOWNORMAL = 1

Private Sub cmdAssociate_Click()
Dim RetVal As Long
    RetVal = ShellExecute(Me.hwnd, _
        vbNullString, _
        "Foo.Txt", _
        vbNullString, _
        "c:\", _
        SW_SHOWNORMAL)
End Sub
```

Figura 16.2 *Possibile avviare un 'applicazione con un file associato, utilizzando la funzione ShellExecute dell'API.*

Per ricercare le costanti e i loro valori per le funzioni API, utilizzate I'API Viewer quando è selezionato Constants *anziché* Declares.

Aspettare che termini un programma avviato tramite la funzione Shell

La funzione Shell esegue un programma esterno. Il primo argomento della funzione è una stringa che rappresenta il file eseguibile; il secondo è un flag costante che indica lo stile della finestra in cui deve essere eseguito il programma (si veda la Tabella 16.1).

Tabella 16.1 *Costanti WindowStyle della funzione Shell.*

Costante	Valore	Descrizione
vbHide	O	La finestra è nascosta e il focus è passato alla finestra nascosta.
vbNormalFocus	1	La finestra ha il focus e viene riFortata alle dimensioni e alla posizione originale.
vbMinimizedFocus	2	La finestra è visualizzata come un'icona con il fuoco.
vbMaximizedFocus	3	La finestra è ingrandita e ha il fuoco.
vbNormalNoFocus	4	La finestra viene riFortata alle dimensioni e alla posizione più recente. Rimane attiva la finestra corrente.
vbMinirnizedNoFocus	6	La finestra è visualizzata come icona. Rimane attiva la finestra corrente.

Se si deve scrivere un'applicazione che possa essere eseguita sia in Windows 3.x che in Windows a 32 bit, è necessario sapere che la funzione Shell funziona in modo diverso in Windows a 32 e a 16 bit. La Shell a 16 bit restituisce l'handle dell'istanza in esecuzione del programma avviato, mentre la Shell a 32 bit restituisce un ID di processo che identifica in modo univoco il programma nella Shell, ma che non *è* sufficiente per accedere al programma. Ogni programma che deve accedere al processo avviato con la funzione Shell deve passare l'ID di processo alla funzione OpenProcess dell'API di Win32 e ottenere un handle al processo.

La funzione Shell a 16 bit

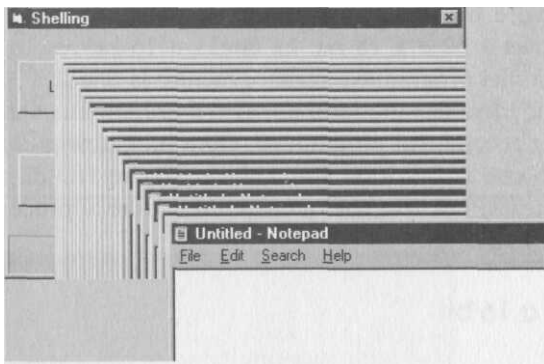
Poiché dalla Shell a 16 bit si ottiene un handle dell'istanza, è possibile utilizzare la funzione Shell e verificare se un programma è terminato mediante la funzione Get - ModuleUsage (non inclusa nell'API di Win32), che restituisce il numero di programmi che utilizzano un modulo:

```
Dim Handle as Integer
Handle = Shell ("NotePad.Exe", vbNormalFocus)
Do While GetModuleUsage (Handle) > 0
   DoEvents
Loop
```

Di seguito sono riportate le dichiarazioni necessarie sotto Win32 per chiamare la funzione Shell e attendere che venga terminato il programma esterno:

```
Private Declare Function OpenProcess Lib "kernel32" _
    (ByVal dwDesiredAccess As Long, _
    ByVal bInheritHandle As Long, _
    ByVal dwProcessId As Long) As Long
Private Declare Function WaitForSingleObject Lib "kernel32" _
    (ByVal hHandle As Long, ByVal dwMilliseconds As Long) As Long
Private Declare Function CloseHandle Lib "kernel32" _
    (ByVal hObject As Long) As Long
Private Const INFINITE = &HFFFF
```

Il codice seguente utilizza l'ID di processo restituito dalla Shell a 32 bit, che viene utilizzato dalla funzione OpenProcess per restituire un handle univoco (vale a dire specifico per una determinata procedura) per il processo avviato con la funzione Shell. Infine viene chiamato WaitForSingleObject con l'handle univoco. WaitForSingleObject entra in uno stato di attesa efficiente che presenta però un aspetto negativo: pur non utilizzando risorse, il programma è tuttavia morto (addirittura non viene aggiornato lo schermo). Eseguendo la funzione Shell dall'ambiente di progettazione di VB mediante questa procedura si causa la chiusura dell'IDE di VB finché viene chiuso il programma avviato con la funzione, perché VB è il vero programma che attende che terminino le operazioni avviate con la funzione Shell (si veda la Figura 16.3).

Figura 16.3 *Chiamando WaitForSingle Object con un handle di processo l'esecuzione riprende solo dopo che è terminato il processo; la sospensione dell'attività è così completa .*

Se si prova a eseguire questo codice, si vedrà che funziona. La casella di messaggio "I'm Back!!!" non viene visualizzata finché non si chiude Notepad. Se si trascina l'applicazione avviata con la funzione Shell (Notepad), si può osservare che lo schermo nell'ambiente di VB non viene ridisegnato.

```
Private Sub cmdShandTer_Click()
    Dim hProg, hProc, RetVal As Long
    Const PROCESS_ALL_ACCESS = 0
    hProg = Shell("Notepad.Exe", vbNormalFocus)  'returns taskID
    'Prende l'handle del processo
    hProc = OpenProcess(PROCESS_ALL_ACCESS, False, hProg)
    'aspetta che il processo termini
```

```
    If hProc <> 0 Then
        RetVal = WaitForSingleObject(hProc, INFINITE)
        CloseHandle hProc
    End If
    MsgBox "I'm Back!!!"
End Sub
```

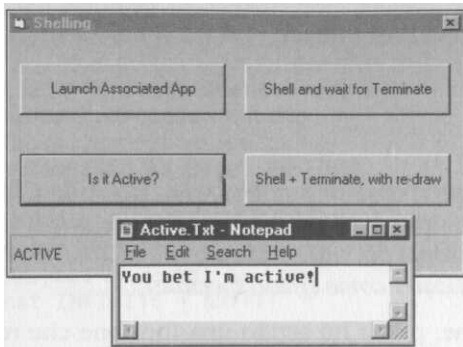
Per quanto mi riguarda, questa sospensione completa dei cicli della CPU al processo che ha effettuato la chiamata non è accettabile. I risultati, in particolare quando viene ridisegnato lo schermo, sono del tutto casuali (in alcune situazioni posso immaginare che lo si possa utilizzare come effetto speciale).

Per modificare questa situazione, prima ho scritto una funzione che restituisce un valore che varia a seconda se il processo chiamato è ancora attivo:

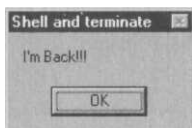
```
Private Declare Function GetExitCodeProcess Lib "kernel32" _
    (ByVal hProcess As Long, lpExitCode As Long) As Long
Private Const INFINITE = &HFFFF

Private Function IsActive(hprog) As Long
    Dim hProc, RetVal As Long
    Const PROCESS_QUERY_INFORMATION = &H400
    Const STILL_ACTIVE = 259
    hProc = OpenProcess(PROCESS_QUERY_INFORMATION, False, hprog)
    If hProc <> 0 Then
        GetExitCodeProcess hProc, RetVal
    End If
    IsActive = (RetVal = STILL_ACTIVE)
    CloseHandle hProc
End Function
```

Nel secondo argomento GetExitCodeProcess restituisce il valore STILL_ACTIVE (definito come uguale a 259) se il processo passato tramite Phandle è ancora attivo. Per verificare questa funzione, al progetto di esempio sono stati aggiunti una barra di stato, un controllo Timer e un pulsante di comando. Quest'ultimo apre l'applicazione esterna, imposta una variabile come identificatore di processo a livello di modulo e attiva il timer. Questo a sua volta chiama la funzione IsActive e, sulla base del valore restituito, aggiorna la barra di stato, come mostrato nella Figura 16.4:

```
Dim IsProg As Long 'Dichiarazione a livello di modulo

Private Sub cmdActive_Click()
   IsProg = Shell("Notepad.exe", vbNormalFocus)
   Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()
    If IsActive(IsProg) Then
        StatusBaM.SimpleText = "ACTIVE"
    Else
        StatusBaM.SimpleText = "NOT ACTIVE"
    End If
End Sub
```

Figura 16.4 *La funzione IsActive determina se un processo esterno è attivo.*

Ora è possibile unire le due parti, il codice Shell e il codice Terminate da una parte e la funzione IsActive dall'altra, in modo da permettere che venga ridisegnato lo schermo dell'applicazione originale utilizzando la dichiarazione DoEvents. L'applicazione originale rimane in attesa in un ciclo Do While che chiama DoEvents, che permette di ridisegnare lo schermo, fin quando l'applicazione avviata con la funzione Shell diventa inattiva.
A questo punto l'esecuzione torna all'applicazione originale, come verificato per mezzo di una casella di messaggio (si veda la Figura 16.5).

```
Private Sub cmdRedraw_Click()
    Dim hprog, hProc, RetVal As Long
    Const PROCESS_ALL_ACCESS = 0
    hprog = Shell("NotePad.Exe", vbNormalFocus) 'returns taskID
    'Prende l'handle del processo
    Do While IsActive(hprog)
        DoEvents
    Loop
    hProc=OpenProcess(PROCESS_ALL_ACCESS,False,hprog)
    'aspetta che il processo termini
    If hProc <> 0 Then
        RetVal = WaitForSingleObject(hProc, INFINITE)
        CloseHandle hProc
   End If
   MsgBox "I'm Back!!!"
End Sub
```

Figura 16.5 *Questa casella di messaggio indica che l'esecuzione è tornata all'applicazione originale.*

Compilazione in pseudocodice e compilazione in codice nativo

In Visual Basic 6 si ha la possibilità di compilare i file eseguibili in codice nativo o in pseudocodice.

Con la versione 6, il codice nativo è diventato più efficiente. In termini generali, vi sono meno motivi per utilizzare lo pseudocodice di VB, motivo per cui la compilazione in codice nativo è diventata l'opzione predefinita.

In questo paragrafo sono discussi brevemente i vantaggi e gli svantaggi dello pseudocodice e del codice nativo, quindi viene spiegato il significato dei commutatori del compilatore di codice nativo.
Come diceva spesso mia nonna, "I vantaggi di uno non sono gli svantaggi dell'altro".
In breve:

- I programmi compilati in pseudocodice sono di dimensioni inferiori.
- In generale, i programmi compilati in codice nativo sono di dimensioni maggiori, ma vengono eseguiti più velocemente.
- Un aspetto negativo della compilazione in pseudocodice è che, oltre al file eseguibile, è necessario distribuire la libreria di esecuzione di Visual Basic (.Dll). Tuttavia, una compensazione parziale è data dal fatto che molti sistemi di destinazione hanno già il runtime di Visual Basic installato.

Secondo Microsoft, "durante test in condizioni reali le applicazioni client tipicamente . . . [spendono] circa il 5% del tempo totale di esecuzione eseguendo lo pseudocodice. Di conseguenza, se il codice nativo fosse istantaneo, il suo utilizzo per questi programmi determinerebbe un miglioramento delle prestazioni al più del 5%".
Questo implica che nella maggior parte delle situazioni, la compilazione in codice nativo consente un miglioramento ridotto delle prestazioni. Naturalmente è difficile valutare la performance, in particolare perché le applicazioni non sempre vengono eseguite nello stesso modo su hardware diverso. Un suggerimento potrebbe essere di verificare le applicazioni nelle quali la velocità dell'esecuzione è critica in entrambe le modalità, utilizzando per la valutazione alcune tecniche descritte più avanti in questo capitolo.
Microsoft afferma che i seguenti tipi specifici di programmi ottengono vantaggi dalla compilazione in codice nativo (se un progetto non è incluso nell'elenco, significa che è probabile che non si ottengano particolari vantaggi dalla compilazione in codice nativo).

Cos'è lo pseudocodice?

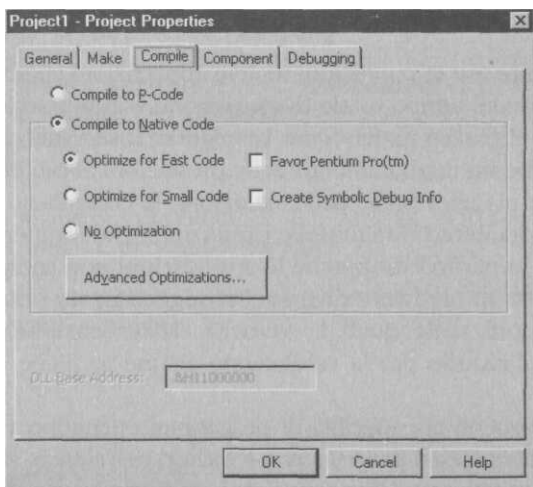
I file eseguibili in pseudocodice sono compilati interamente in un linguaggio suddiviso in token. Lo pseudocodice compilato in questo modo viene quindi trasferito (interpretato) in codice macchina dal modulo di esecuzione di Visual Basic. È possibile pensare allo pseudocodice di Visual Basic come a un passaggio intermedio tra il codice di alto livello di Visual Basic e quello di basso livello della macchina.

- I programmi che eseguono molte operazioni primitive su variabili codificate, non stringa (ad esempio calcoli finanziari complessi o generatori di frattali)
- I programmi complessi dal punto di vista dell'elaborazione
- i programmi che spostano frequentemente bit e byte nelle strutture di dati locali

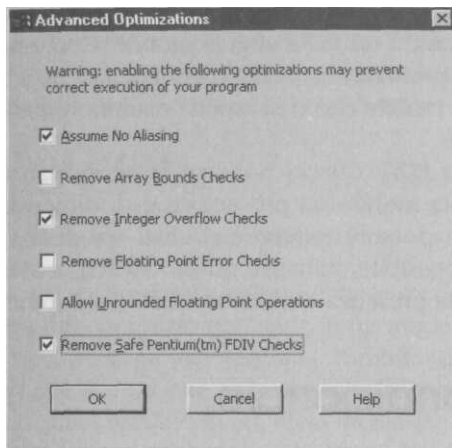
Al contrario, i programmi che spendono molto tempo chiamando l'API di Windows, chiamando oggetti ActiveX, manipolando stringhe o utilizzando funzioni nella libreria di VBA non ottengono un grosso miglioramento dalla compilazione in codice nativo.

Switch del codice nativo

La scheda *Compile* della finestra di dialogo *Project Properties* è utilizzata per selezionare la compilazione in codice nativo al posto della compilazione predefinita in pseudocodice. La scheda *Compile,* mostrata nella Figura 16.6, si apre scegliendo *Project Properties* dal menu *Project* oppure scegliendo *Options* nella finestra di dialogo *Make Exe.*

Figura 16.6
La scheda Compile della finestra di dialogo Project Properties viene utilizzata per selezionare le opzioni di compilazione.

La scheda *Compile* include diverse opzioni (si veda la Figura 16.6). È possibile scegliere se fare in modo che il codice nativo eseguibile sia ottimizzato in funzione della velocità o delle dimensioni oppure che non sia ottimizzato. Se si seleziona *Favor Pentium Pro,* viene generato codice che viene eseguito più velocemente su processori Pentium Pro (ma più lentamente sulle altre CPU). Selezionando *Create Symbolic Debug Info* si fa in modo che vengano generate informazioni sul debugging che possono essere visualizzate nell'ambiente di Visual C++.

Se si sceglie *Advanced Optimizations* si apre la finestra di dialogo *Advanced Optimizations,* mostrata nella Figura 16.7. Queste ottimizzazioni non sono sicure dal punto di vista dell'elaborazione: se si sa cosa si sta facendo, selezionando queste opzioni si può migliorare la velocità degli eseguibili, ma si corre il rischio di creare codice non stabile.

Figura 16.7
Nella finestra di dialogo Advanced Optimizations si possono selezionare opzioni per la compilazione in codice nativo che possono aumentare la velocità del codice, ma che sono potenzialmente non sicure.

La maggior parte di queste opzioni comForta la rimozione di verifiche interne del codice. Personalmente preferisco essere protetto dalle mie stesse disattenzioni, in particolare quando è necessario lavorare molto per terminare un programma nei tempi previsti.

Aliasing significa chiamare la stessa variabile con più di un nome. Questo avviene, ad esempio, quando una variabile viene passata a una routine e quindi viene utilizzata da questa routine come variabile globale e come parametro.

Se non si è sicuri se in un progetto è presente l'aliasing, selezionando *Assume No Aliasing* si permette al compilatore di applicare delle ottimizzazioni che diversamente non potrebbe utilizzare, ad esempio memorizzare le variabili in registri ed eseguire ottimizzazioni dei cicli. Selezionando *Remove Array Bound Checks* si disattiva la verifica degli errori dei limiti degli array e il controllo della correttezza delle dimensioni degli array.

Microsoft ha avvisato che selezionando l'opzione Remove Array Bound Checks *"si può aumentare la velocità della manipolazione degli array, ma si può accedere a locazioni errate della memoria, causando un comFortamento inaspettato o un 'interruzione delprogramma ".*

In base alle impostazioni predefmite, in Visual Basic ogni valore interno viene controllato per vedere se rientra nell'intervallo dei valori possibili. Selezionando *Remove Integer Overflow Checks* si disattiva questo controllo, cosa che può aumentare la velocità dei calcoli degli interi. Tuttavia, se vi sono delle variabili intere al di fuori dell'intervallo, non si riceve alcun messaggio di errore e probabilmente si otterranno risultati errati.

In modo simile viene effettuato un controllo di tutti i calcoli dei dati a virgola mobile, singola e doppia, per assicurarsi che il valore sia all'interno della gamma e che non si eseguano divisioni per zero o altre operazioni non valide. Se si seleziona *Remove Floating Point Error Cbecks,* viene disattivata la verifica di questo errore. Anche in questo caso, la velocità aumenta, ma se qualcosa va storto non si riceve alcun messaggio, bensì solo risultati anomali.
L'opzione *Allow Unrounded Floating Point Operations,* se selezionata, permette a VB di gestire in modo più efficiente i calcoli a virgola mobile. Come nel caso della finestra di dialogo *Advanced Optimizations,* una conseguenza negativa è che il confronto di due valori a virgola mobile che ci si aspetta risultino uguali invece può essere valutato come non uguale.
L'opzione *Remove Safe Pentium FDIV Checks* rimuove la verifica in modo che il codice per la divisione in virgola mobile sia più veloce e di dimensioni inferiori. Selezionando questa opzione si possono ottenere risultati sbagliati su processori Pentium con il difetto FDIV (è possibile utilizzare questo switch, assieme a un calcolo appropriato, per verificare la presenza del difetto FDIV dei Pentium).

Compilazione condizionale

È possibile controllare quale parte del progetto viene compilata utilizzando le dichiarazioni `#If. . .#ElseIf...#Else...#End If`. I blocchi di compilazione condizionale utilizzano il valore di costanti condizionali del compilatore (discusse di seguito) per determinare quali blocchi di codice sono inclusi nel programma eseguibile finale che viene compilato. Il codice ignorato a causa della compilazione condizionale non aumenta le dimensioni dell'eseguibile e non consuma risorse. Si può dire che la compilazione condizionale permette di creare commenti per escludere blocchi di codice in modo semplice e selettivo. Un utilizzo comune consiste nell'includere codice con costanti condizionali per il debugging.
Un utilizzo imFortante della compilazione condizionale è la gestione delle differenze tra piattaforme operative. Ad esempio, potrebbero esservi delle differenze tra Windows 95/98 e Windows NT che devono essere gestite in modo condizionale, se si intende includere queste funzionalità nel codice ed eseguirle su entrambe le piattaforme.
Un altro utilizzo della compilazione condizionale consiste nell'includere un file di risorse esterne, contenente letterali stringa, bitmap e così via, sulla base di una costante condizionale. Si tratta di una tecnica eccellente per intemazionalizzare le applicazioni (per ulteriori dettagli si veda il paragrafo "File di risorse esterni" più avanti in questo capitolo).
È anche possibile utilizzare la compilazione condizionale per includere, o escludere, codice utilizzato per il debugging o per misurare la velocità di parti di un'applicazione (si veda il paragrafo "Misurazione della velocità" più avanti in questo capitolo).

Costanti condizionali

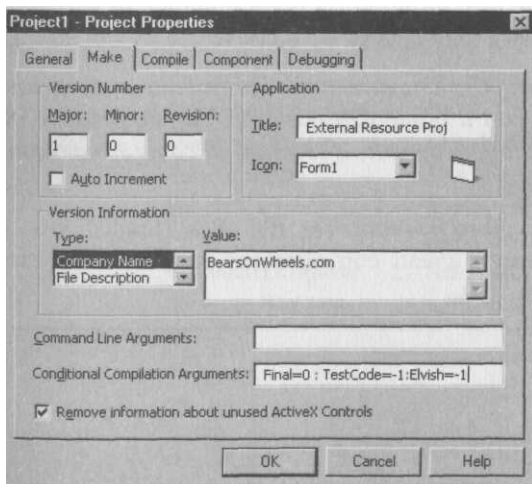
Non è possibile mischiare e abbinare costanti condizionali del compilatore con codice diverso. In altre parole, è possibile utilizzarle solo per la compilazione condizionale (ad esempio non è possibile impostare una costante condizionale sulla base della valutazione di una normale istruzione If).
Le costanti condizionali sono True quando l'espressione a loro assegnata viene valutata come -1, diversamente sono False. È possibile assegnare a una costante condizionale altre costanti condizionali e operatori aritmetici e logici (ad eccezione di Is). Ad esempio:

```
#Const Final = 0 'False
#Const TestCode = -1 'True
#Const Elvish = -1 'True
#Const English = Not #Elvish 'False
```

Si possono creare costanti personalizzate in tre modi. Innanzi tutto è possibile utilizzare la direttiva del compilatore #Const in un modulo. Indipendentemente da dove si trova questa istruzione nel modulo, l'ambito della costante condizionale è a livello di modulo. Si noti che le costanti condizionali dichiarate in questo modo sono *sempre* locali al modulo in cui sono dichiarate.
È possibile dichiarare una costante condizionale globale rispetto al progetto nella riga di comando utilizzando il commutatore /d o la scheda *Make* della finestra di dialogo Project Properties. Ad esempio, è possibile utilizzare il commutatore /make per compilare un progetto di VB con costanti condizionali dalla riga di comando:

```
C:\Vb98\Vb6.Exe /make Project1 .Vbp /d TestCode=-1 : Elvish=-1
```

Il modo più comune per immettere costanti condizionali a livello di progetto consiste nell'utilizzare la scheda *Make* della finestra di dialogo *Project Properties* (che può essere aperta anche scegliendo *Options* nella finestra di dialogo *Make Project),* come mostrato nella Figura 16.8.

Figura 16.8
È possibile impostare le costanti condizionali del compilatore utilizzando la scheda Make della finestra di dialogo Project Properties.

File di risorse esterni

Supponete di dover preparare versioni di un software per molte lingue diverse. Certamente sarebbe preferibile se ci si potesse concentrare sulla logica del programma e distribuire un elenco di letterali stringa (e di immagini) necessari per tradurre ciò che viene visualizzato nelle diverse lingue, che potrebbero essere compilati da qualcun altro. Ad esempio si potrebbe far tradurre 1 = "My Test Program" (la didascalia del form Main), 2 = "OK" e così via nelle diverse lingue.

Tutto questo può essere fatto utilizzando la compilazione condizionale e file di risorse esterne. Questi risultano particolarmente utili quando è necessario semplicemente preparare numerose versioni di un programma con visualizzazioni leggermente diverse, anche se tutte le versioni sono nella stessa lingua.

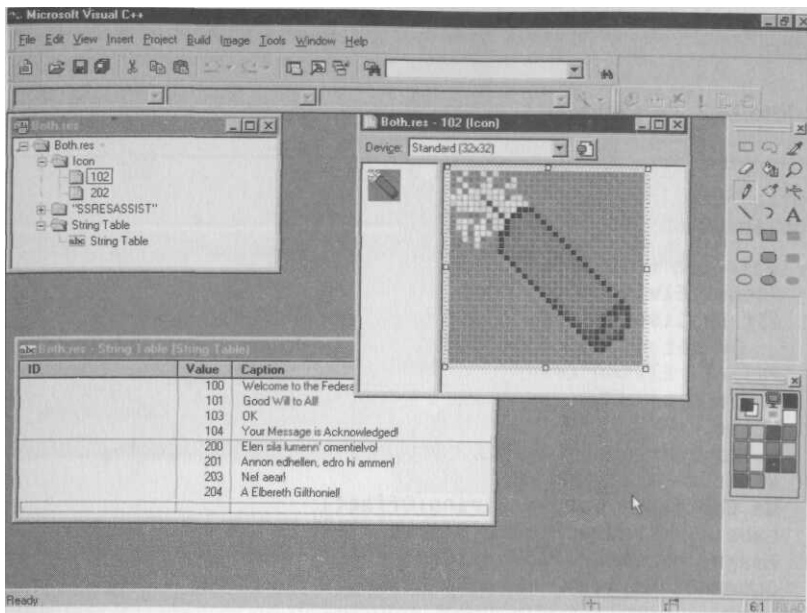
Un vantaggio che si ottiene con questo schema, vale a dire separando gli elementi dello schermo quali i testi e le immagini dal programma stesso, è che i file di risorse possono essere dati ad altre persone, senza compromettere la sicurezza dei file di progetto.

Per aggiungere a un progetto un file di risorse esterne (.Res) compilato, scegliere *Add File* dal menu *Project.* In un progetto è possibile caricare un solo file .Res alla volta (se si cerca di caricarne un altro si riceve un messaggio di errore). Ciò significa che vi sono due modi per gestire le risorse esterne:

- È possibile avere un file di risorse per ogni lingua. I numeri di identificazione interni degli elementi devono essere uguali. Ad esempio, se in English.Res l'ID della stringa "OK" è 101, in German.Res l'ID di "Ja" deve essere anch'esso 101. In un progetto può essere caricato solo uno di questi file alla volta.
- È possibile inserire un unico grande file di risorse ordinate secondo la lingua e differenziate l'una dall'altra da un numero specifico. Quindi si utilizza la compilazione condizionale per compilare solo le risorse necessarie nel programma. Se si utilizza questo approccio, ci si deve assicurare che vi sia sufficiente spazio tra i gruppi di lingua quando si assegnano i numeri di identificazione alle risorse, in modo da poter aggiungere successivamente nuòvi elementi in base alle necessità.

Per rendere tutto questo un po' meno astratto, ho preparato un programma di esempio, salvato nel CD-ROM con il nome External. Vbp. Questo progetto visualizza un form con elementi in due lingue diverse, a seconda di come è impostato il flag della compilazione condizionale.

Nel CD-ROM allegato al libro vi sono tre file .Res (English.Res, Elvish.Res e Both.Res) che possono essere creati con numerosi strumenti, ad esempio Visual C++, mostrato nella Figura 16.9.

Figura 16.9
Perpreparare un file di risorse esterne èpossibile utilizare Visual C++oaltri strumenti per la modifica delle risorse.

In VB6 è inclusa una versione ridotta di Resource Editar. Per attivarla si utilizza Add-In Manager, descritto nel Capitolo 29.

Nell'applicazione di esempio è stato caricato Both.Res e viene utilizzata la compilazione condizionale con un offset per aggiungere al progetto le risorse appropriate. Ho preparato Both.Res sulla base delle informazioni riFortate nella Tabella 16.2, con valori per English che partono da 100 e valori per Elvish che partono da 200.

Tabella 16.2 *Informazioni da includere nel file di risorse (.Res) compilato.*

Valore(ID)	Funzione	Contenuto del letterale stringa
100	Form.Caption	Welcome to the Federation!
101	Label1.Caption	Good Will to Ali
102	Imagel.Picture	Icona petardo
103	Command1 .Caption	OK
104	Message Box text	Your Message is Acknowledged!
200	Form.Caption	Elen sila lumenn' omentielvo!
201	Label1.Caption	Annon edhellen, edro hi ammen!
202	Imagel.Picture	Icona dell'albero di Natale
203	Comman1.Caption	Nef aear!
204	Message Box text	A Elbereth Gilthoniel!

Il Listato 16.3 include il codice necessario per caricare la versione Elvish del progetto:

Listato 16.3 *Caricamento di unform mediante costanti condizionali e un file di risorse esterne.*

```
Option Explicit
Public Offset As Integer

Private Sub Form_Load()
   #Const Elvish = -1 'True
   #If English Then
      Offset = 100
   #ElseIf Elvish Then
      Offset = 200
   #Else
      MsgBox "Tentativo di caricare una lingua ignota!"
   #End If
   Me.Caption = LoadResString(Offset)
   Label1 = LoadResString(Offset + 1)
   Image1 = LoadResPicture(Offset + 2, vbResIcon)
   Command1.Caption = LoadResString(Offset + 3)
End Sub

Private Sub Command1_Click()
   MsgBox LoadResString(Offset + 4), _
      vbInformation,
      LoadResString(Offset)
End Sub
```

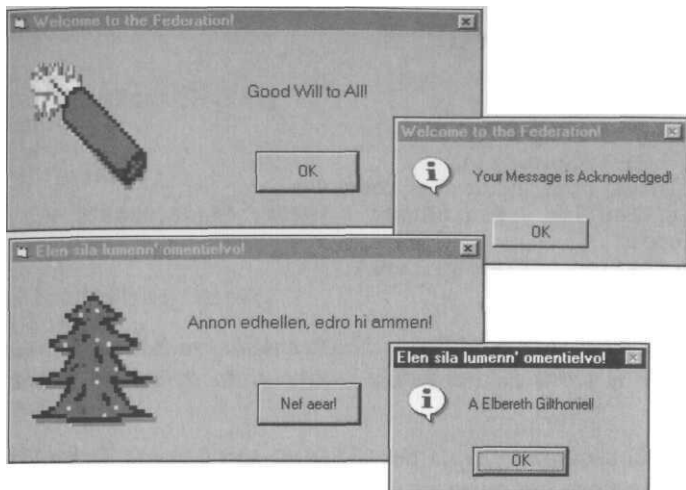
Per passare alla versione inglese, basta cambiare la dichiarazione #Const:

```
#Const English = -1 'True
```

In alternativa è possibile impostare il valore della costante nella scheda Make della finestra di dialogo Project Properties.
È possibile eseguire entrambe le versioni English e Elvish contemForaneamente in copie diverse dell'ambiente di sviluppo di VB, ottenendo i risultati mostrati nella Figura 16.10.

Ottimizzazione

Quando i programmatori parlano di "ottimizzazione", normalmente intendono "ottimizzazione in funzione della velocità". Infatti in un programma è possibile cercare di ottimizzare la velocità, la velocità apparente, la visualizzazione, la visualizzazione apparente o il consumo di memoria. Cercare di ottimizzare tutti questi aspetti non è contraddittorio di per sé, ma a volte i risultati ottenuti potrebbero non essere dei migliori. Potrebbe essere necessario decidere cosa è più imFortante e impostare di conseguenza le proprie priorità per l'ottimizzazione.

Figura 16-10
I file di risorse esterne compilati in modo condizionale costituiscono un modo eccellente per localizzare le applicazioni

Parleremo prima dell'ottimizzazione in funzione della velocità, perché è quella più semplice. Senza troppe difficoltà è possibile quantificare in modo attendibile il tempo che consuma il codice di un programma. Prima di poter veramente ottimizzare il codice in funzione della velocità, è necessario essere in grado di misurare quanto veloci sono le diverse parti di un programma. Senza identificare i punti in cui si hanno degli strozzamenti, è impossibile determinare quali parti del codice devono essere messe a punto. Non vi è motivo di ottimizzare procedure che non lo richiedono. Il tempo effettivamente speso per l'elaborazione può non essere indicativo e pertanto, senza strumenti di misurazione analitica, non è possibile essere sicuri del tempo effettivamente utilizzato dalle diverse procedure.

Misurazione in funzione della velocità

Le procedure di misurazione del tempo presentate nel seguito sono salvate nel CD-ROM allegato al libro in un modulo chiamato Time.Bas. Vi sono in particolare due procedure, StartTimer *e* StopTimer, *da utilizzare come segnalibro in qualsiasi codice che si desidera misurare.*

Queste semplici procedure utilizzano la funzione GetTickCount dell'API della libreria di Kernel32. GetTickCount fornisce una stima più accurata del tempo trascorso rispetto alle funzioni di VB e misura il tempo in millesimi di secondo da quando è stato avviato Windows.

Per gli appassionati può essere interessante sapere che il timer interno di Windows torna a O dopo che è stato in funzione continuamente per circa 49, 7 giorni.

Il modulo Time.Bas è il seguente:

```
Option Explicit
Declare Function GetTickCount Lib "kernel32" () As Long
Public BeginTime As Long
```

```
Public Sub StartTimer()
   BeginTime = GetTickCount()
End Sub

Public Sub StopTimer()
   Dim EndTime As Long
   EndTime = GetTickCount ()
   MsgBox "Total time used: " + vbCrLf + _
      Format((EndTime - BeginTime) / 1000#, "###0.0000") +_
      " Seconds",_
      vbInformation, "Elapsed Time"
End Sub
```

Se lo si desidera, èpossibile modificare il codice nella procedura StopTimer *in modo da visualizzare il pannello* Immediate *utilizzando il metodo* Print *dell'oggetto* Debug.

L'applicazione di esempio, salvata nel CD-ROM con il nome Timer.Vbp, utilizza le procedure Timer Code per misurare il tempo utilizzato da diverse operazioni:

- iterare 100.000 volte con un ciclo For... Next. Tempo trascorso: 0,009 secondi.
- iterare 100.000 volte con un ciclo Do While. Tempo trascorso: 0,066 secondi.
- aggiungere 10.000 stringhe (la stringa di esempio è: "Hacker") a un array con For. . .Next. Tempo trascorso: 0,06 secondi.
- aggiungere 10.000 stringhe (anche in questo caso è utilizzata la stringa "Hacker") a un array ridimensionato dinamicamente ogni volta. Tempo trascorso: 0,076 secondi.
- aggiungere 10.000 stringhe di un carattere alfabetico generato casualmente a un array ridimensionato dinamicamente. Tempo trascorso: 0,37 secondi.

Questi tempi si basano sull'esecuzione del programma nell'ambiente di VB (se si esegue la versione compilata del programma all'esterno dell'ambiente di VB si ottengono prestazioni migliori). Inoltre, naturalmente, dipendono dal sistema (in questo caso una CPU Pentium Pro da 200 megahertz). I rapForti relativi dovrebbero tuttavia rimane più o meno invariati in qualsiasi sistema. È imFortante ricordare che, in particolare nei sistemi operativi multithreaded, i valori assoluti dipendono dallo stato del computer e da quali altre operazioni sta eseguendo contemForaneamente. Certamente da questi test di esempio è possibile giungere alla conclusione che ridimensionando dinamicamente gli array le prestazioni peggiorano e che utilizzando il generatore casuale di VB (la funzione Rnd) si provocano dei veri rallentamenti della performance. Il Listato 16.4 contiene il codice per le procedure di verifica.

listato 16.4 *Misurazione del tempo trascorso.*

```
Option Explicit
Private Sub cmdForNext_Click()
Dim i As Long
   StartTimer
   For i = 1 To 100000
   Next i
   StopTimer
End Sub

Private Sub cmddowhile_Click()
   Dim x As Long
   StartTimer
      Do While x < 100000
         x = x + 1
      Loop
   StopTimer
End Sub

Private Sub cmdBigStringArray_Click()
   Dim i As Integer
   Dim Contains(10000) As String
      StartTimer
         For i = 0 To 9999
            Contains(i) = "Hacker"
         Next i
      StopTimer
End Sub

Private Sub cmdDyna_Click()
   Dim i As Integer
   Dim Contains() As String
   StartTimer
   Fon i = 0 To 9999
      ReDim Preserve Contains(i)
      Contains(i) = "Hacker"
   Next i
   StopTimer
End Sub

Private Sub cmdRandom_Click()
   Dim i As Integer
   Dim Contains() As String
   StartTimer
   For i = 0 To 9999
      ReDim Preserve Contains(i)
      Contains(i) = Asc(Rnd * 26)
   Next i
   StopTimer
End Sub
```

Se si esegue il progetto di esempio, una finestra di messaggio comunica il tempo utilizzato dalla routine selezionata, come mostrato nella Figura 16.11.

Figura 16.11
È semplice misurare il tempo utilizzato dal codice di programma.

Ottimizzazione in funzione della velocità

È possibile utilizzare le tecniche e le regole pratiche presentate in questo paragrafo per aumentare la velocità effettiva delle applicazioni.

- Evitare le variabili varianti. Utilizzare le variabili esplicite tramite la dichiarazione `Option Explicit` e fare attenzione alle conversioni implicite delle variabili.

Per esempio, la dichiarazione `Dim I,J,K As Integer` *crea una variabile intera (K) e due varianti (I e J). È facile fraintendere la sintassi, perché si può pensare di aver digitato esplicitamente delle variabili, mentre in realtà alcune sono varianti.*

- Se possibile, utilizzare variabili intere e la matematica con gli interi. Byte, Integer e Long sono i tipi di dati nativi dei processori Intel e le operazioni con questi dati sono sorprendentemente veloci. Utilizzando la matematica con gli interi è possibile arrivare a una quantità incredibile di operazioni.
- Separare le applicazioni di grandi dimensioni in numerosi componenti ActiveX separati che sfruttano il multitasking preemptive di Windows a 32 bit. È possibile utilizzare metodi di automazione remota per delegare alcuni compiti di elaborazione alla CPU remota, se ciò è appropriato nel contesto dell'applicazione che si sta creando.
- Evitare di copiare stringhe, se non è necessario.
- Per le operazioni di I/O dei file, utilizzare l'accesso binario (al posto di quello testuale o casuale).
- Sostituire le matrici (array) con le collezioni, se in questo modo si ottengono tempi più brevi, e utilizzare `For Each` sulle collezioni anziché gli

indici. Se è possibile associare una chiave unica a ogni elemento della collezione, questa è l'opzione più veloce. Le matrici invece vanno meglio per quanto concerne l'accesso sequenziale quando si deve operare su tutti gli elementi nella collezione o nella matrice. Se possibile, evitare di utilizzare il parametro Before o After quando si aggiungono elementi a una collezione, perché è necessario più tempo per aggiungere un elemento a una collezione, se deve anche essere posizionato.

Se si utilizza più volte una proprietà, è meglio leggerla in una variabile e utilizzare quest'ultima. Per esempio, il codice:

```
For I = 1 to 5
    optWhich.Caption = Command1.Caption
Next I
```

è più lento di

```
Saying = Command1.Caption
For I = 1 to 5
    optWhich.Caption = Saying
Next I
```

Ottimizzazione in funzione della velocità apparente

L'ottimizzazione della velocità apparente copre due argomenti correlati: aumentare la velocità di visualizzazione sullo schermo e impostare gli aspetti di un programma in modo che questo sembri essere più veloce. Di seguito sono presentati alcuni suggerimenti relativi a entrambi questi argomenti.

- Non eseguire operazioni lunghe, ad esempio l'inizializzazione di un database, nell'evento Load di un form. Poiché l'operazione ritarda la visualizzazione del form sullo schermo, l'utente percepisce l'applicazione come se fosse lenta.
- Avviare le applicazioni utilizzando una schermata di avvio come quella descritta precedentemente in questo capitolo, in modo da nascondere le operazioni iniziali che sono piuttosto lunghe.
- Utilizzare i controlli Image al posto dei controlli Picture. Questi ultimi utilizzano molte più risorse, in quanto sono controlli di finestra, al contrario dei controlli Image. Se si deve visualizzare un'immagine su cui è possibile fare clic e che magari può essere trascinata e rilasciata, il controllo Image è più che sufficiente.

  I controlli Label consumano una quantità molto ridotta di risorse (non sono di finestra).

  E possibile utilizzare un'etichetta nascosta per memorizzare il testo di cui si avrà bisogno.

- È anche possibile utilizzare la proprietà DragIcon del controllo Label per memorizzare le icone necessarie in un progetto.
- Utilizzare il metodo Line al posto del metodo PSet per disegnare gli oggetti.
- Impostare *ClipControls* su False nella finestra *Properties* per i form e i controlli di un progetto, in modo da ridurre il tempo necessario per ridisegnare lo schermo. Se si imposta *ClipControls* su True, i metodi grafici negli eventi Paint ridisegnano l'intero oggetto, mentre se lo si imposta su False vengono ridisegnate solo le aree modificate.
- Assicurarsi che i controlli siano nascosti (impostandone la proprietà Visible su False) quando si applicano diversi cambiamenti che ne coinvolgono le proprietà. In questo modo il controllo viene ridisegnato una sola volta, quando la sua proprietà Vislble è impostata su True, anziché ogni volta che viene applicato un cambiamento a una proprietà.
- Fare delle prove con l'impostazione della proprietà AutoRedraw, che permette di ridisegnare automaticamente un controllo Form o PictureBox. A volte, disattivando AutoRedraw si libera una considerevole quantità di memoria.
- Per ridurre il tempo apparente di caricamento, tenere i form nascosti, ma caricati finché se ne ha bisogno (anche se, ovviamente, in questo modo si consuma più memoria rispetto a caricare i form solo quando servono effettivamente).
- Comunicare con gli utenti. Utilizzare degli indicatori di avanzamento, ad esempio il controllo ProgressBar e visualizzare messaggi che indicano cosa sta avvenendo. Permettere agli utenti di arrestare i processi troppo lunghi.
- Accedere meno volte possibile ai dati memorizzati su disco. Se un programma deve leggere delle informazioni, è consigliabile fare in modo che ne legga il più possibile. Il tempo extra necessario per leggere più informazioni non è avvertibile e i dati aggiuntivi sono immediatamente disponibili quando si rendono necessari. Naturalmente è necessario implementare questa strategia in modo intelligente, così da non perdere tempo a leggere dati che non verranno mai utilizzati.

Riduzione del consumo di memoria

Vi sono numerosi modi per ridurre lo spazio che un'applicazione occupa nella memoria. Spesso tuttavia è necessario decidere quale obiettivo deve avere la priorità. Se si nascondono i form si consuma memoria, ma se ne riduce il tempo di caricamento. Molte decisioni relative allo sviluppo di applicazioni implicano questo tipo di compensazione. Ogni decisione deve essere presa sulla base della situazione specifica. I suggerimenti riFortati in questo paragrafo aiutano a evitare di sprecare memoria inutilmente.

- Scaricare completamente i form. Quando delle variabili contengono istanze di form che non sono più necessarie, impostarle su Nothing. Per esempio:

```
Dim X As New Form1
'Fa qualcosa con X

Unload X
Set X = Nothing
```

È consigliabile assegnare Nothing alle variabili oggetto istanziate, anche quando la variabile rappresenta un form che è stato scaricato con la dichiarazione Unload.

Riciclare lo spazio assegnato a variabili stringa che non sono più necessarie assegnando alla variabile una stringa vuota:

```
MyString = "" 'Recupera lo spazio
```

- Nonostante lo spazio utilizzato da variabili stringa locali e da altre variabili venga riciclato automaticamente quando la variabile esce dal suo ambito, le variabili globali continuano ad avere visibilità finché il progetto è in esecuzione. Se si devono utilizzare stringhe globali in un progetto, assegnare loro delle stringhe vuote quando non sono più necessarie, in modo da ridurre il consumo di memoria.
- Se un programma non ha più bisogno di una matrice dinamica, riciclare lo spazio utilizzato dalla matrice per mezzo della dichiarazione Erase in modo da eliminare in modo sicuro la matrice:

```
Erase MyArray
```

Erase rimuove completamente lo spazio di memorizzazione allocato per la matrice dinamica. Utilizzare ReDim Preserve per specificare il limite superiore della matrice con il valore più basso, se è necessario mantenere alcuni elementi della matrice, riducendone però le dimensioni in modo da utilizzare meno spazio nella memoria. Evitare l'utilizzo di variabili varianti che utilizzano molta memoria e che occupano più spazio delle variabili di tipo fisso che contengono le stesse informazioni. Naturalmente, in alcune circostanze l'utilizzo delle varianti è appropriato, ad esempio per evitare possibili errori di overflow.

- Eliminare il codice inutilizzato. Rimuovere tutto il codice a cui non viene fatto riferimento durante l'esecuzione del progetto, incluse le variabili non utilizzate.

Riciclare la memoria utilizzata dalle immagini nei controlli Image e Picture. Se il controllo non "verrà più utilizzato, non nasconderlo, ma eliminare l'immagine dalla memoria. Vi sono tre modi per farlo:

```
Image1.Picture=LoadPicture()
```

oppure

```
Set Image1.Picture = Nothing
```

oppure, per i controlli dei form e delle immagini, che hanno una proprietà AutoRedraw,

```
Form1.AutoRedraw = True
Form1.Clear
Form1.AutoRedraw = False
```

Gli elementi grafici possono essere condivisi fra controlli Picture e controlli Image, pertanto non vi è motivo di caricare un elemento grafico più di una volta. Ad esempio:

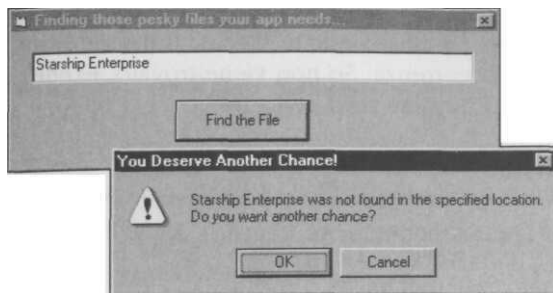
```
MyPic = LoadPicture("C:\Windows\Arches.Bmp")
Image1 = MyPic
Picture1 = MyPic
```

Se durante la progettazione si carica un'immagine in più controlli o forni, con ognuno di essi viene salvata una copia dell'immagine. Invece si dovrebbe caricarla una volta sola, risparmiando memoria e il tempo necessario per diverse letture del disco.

È ancora meglio se si evita completamente di memorizzare le immagini durante la progettazione. Infatti è possibile memorizzarle come risorse (si veda il paragrafo "File di risorse esterne" nella parte precedente di questo capitolo) e caricarle quando necessario durante l'esecuzione tramite la funzione LoadResPicture. Se non si utilizzano contemForaneamente tutte le immagini e tutti i controlli in un forni, in questo modo si risparmia memoria e si può anche velocizzare il caricamento dei form, in quanto non è necessario che tutte le immagini vengano caricate prima che possa essere visualizzato il form.

Ricerca di file sul disco

Il programma dimostrativo di questo paragrafo mostra come si può vedere se un file si trova nella directory o nel percorso dell'applicazione. L'idea alla base di questo programma si richiama al progetto della schermata di *avvio* all'inizio di questo capitolo: occuparsi delle operazioni richieste dal programma, coprendole con una schermata di avvio o con un'animazione.
Le operazioni in questione riguardano i file necessari all'applicazione, ad esempio i file database, i file delle chiavi di cifratura e così via. Nelle applicazione complete, il nome e la posizione di questi file vengono memorizzati nel Registro (si faccia riferimento al Capitolo 10); quindi si utilizzano routine simili a quella mostrata in questo capitolo per verificare che i file siano realmente presenti in quella posizione. Si potrebbe addirittura utilizzare una struttura espandibile (una collezione o una matricedinamica), nel caso in cui non si conosca il numero di file da verificare. Dopo aver controllato la posizione dei file, si può assegnare la loro posizione a variabili globali (o agli elementi di una struttura globale).
Il senso di questa procedura si manifesta quando il file *non* viene trovato. Il codice di esempio cerca prima automaticamente di individuare il file nella directory dell'applicazione o in qualsiasi directory nel percorso. Se il file non viene trovato, l'utente può indicare la posizione del file (si veda la Figura 16.12). Se necessario, ad esempio, l'utente può copiare il file nel disco fisso.

Figura 16.12 *Il programma di dimostrazione permette all'utente di determinare la posizione di file importanti.*

Ilprogetto, salvato nel CD-ROM con il nome FindFile. Vbp, utilizza le routine dei file in un modulo chiamato Files.Bas.

Come mostrato nel Listato 16.5, la funzione Exists utilizza la funzione VB FileLen per determinare se un file esiste:

Listato 16.5 *Verifica dell'esistenza di un file.*

```
Public Function Exists(F As String) As Boolean
    Controlla se un certo file esiste
Dim X As Long
   On Error Resume Next
   X = FileLen(F)
   If X Then
      Exists = True
   Else
      Exists = False
   End If
End Function
```

StripPath "decapita" il percorso e restituisce il semplice nome di file, come mostrato nel Listato 16.6.

Listato 16.6 *Restituzione di un nome di file senza percorso.*

```
Public Function StripPath(T As String) As String
    Decapita il percorso, restituisce il nome del file
   On Error Resume Next
   Dim X As Integer
   Dim et As Integer
   StripPath = T
   X = InStr(T, "\")
   Do While X
      et = X
      X = InStr(ct + 1, T, "\")
   Loop
   If et > 0 Then StripPath = Mid(T, et + 1 )
End Function
```

FindFile Ricerca il nome di file che è stato passato, prima nella directory dell'applicazione e quindi nel percorso. Se il file viene trovato, la funzione restituisce la posizione completa della prima occorrenza. Se non viene trovato, FindFile restituisce una stringa vuota, come si può vedere nel Listato 16.7.

Listato 16.7 *Ricerca di un file.*

```
Public Function FindFile(SearchFile As String) As String
    Dim Path As String, CurrentDir As String
    Dim found As Integer, semicolon As Integer
    On Error GoTo ErrHandle
    CurrentDir = App.Path
    If Right(CurrentDir, 1) <> "\" Then _
        CurrentDir = CurrentDir + "\"
    found = Dir(CurrentDir & SearchFile) <> ""
    If Not found Then
        Path=Environ("PATH")
        If Path <> "" Then
            If Right(Path, 1) <> ";" Then Path = Path + ";"
                semicolon = InStr(Path, ";")
                Do
                    CurrentDir = Left(Path, semicolon - 1)
                    If Right(CurrentDir, 1) <> "\" Then _
                        CurrentDir = CurrentDir + "\"
                    found = Dir(CurrentDir & SearchFile) <> ""
                    Path = Right(Path, Len(Path) - semicolon)
                    semicolon = InStr(Path, ";")
                Loop While ((semicolon <> 0) And Not found)
            End If
    End If
    If found Then
        FindFile = CurrentDir & SearchFile
    Else
        FindFile = ""
    End If
    Exit Function
ErrHandle:
    MsgBox "Error Number: " + Str(Err.Number) + _
        ";Description: " + Err.Description
    ResumeNext
End Function
```

Questa funzione relativamente complessa utilizza la funzione Environ di VB per restituire una stringa con il percorso corrente e quindi analizza il percorso, suddividendolo in directory e utilizzando quale delimitatore tra le dichiarazioni del percorso un punto e virgola.
Il codice di esempio Find File chiama prima la funzione Exists per vedere se il testo immesso in txtFile è un file esistente:

```
Private Sub cmdFind_Click()
    Dim FileOut As String, FileIn As String, RetVal As Long
    FileIn = Trim(txtFile.Text)
```

```
If Not Exists(FileIn) Then
```

Se viene trovato il file come è stato specificato, l'esecuzione termina con la clausola Else alla fine della procedura. Se invece il file non viene trovato:

```
FileOut = FindFile(StripPath(FileIn))
If FileOut = "" Then
   RetVal = MsgBox(FileIn + " non è stato trovato in" _
      + "posizione specificata." + _
      vbCrLf + "Vuoi un'altra possibilità?", _
      vbExclamation + vbOKCancel, _
      "Hai diritto a un'altra possibilità!")
   If RetVal = vbOK Then RepairIt FileIn
   Exit Sub
End If
```

FindFile viene chiamata usando come argomento la versione "decapitata" del percorso del file immesso. Se FindFile restituisce una stringa vuota, significa che nella directory dell'applicazione o nel percorso non è stato trovato un file con quel nome e all'utente viene data la possibilità di modificare la situazione nella procedura ReachIt (descritta di seguito). La fine della sottoroutine cmdFind è:

```
   Else
      FIleOut = FileIn
   End If
   MsgBox "File Found: " + fileout, vbInformation, _
      "Global File Variable assignment! "
   txtFile.Text = fileout
End Sub
```

Il codice RepairIt nel programma dimostrativo è abbastanza scarno, ma può essere elaborato ulteriormente e includere qualsiasi tipo di istruzione condizionale.

Selezione del testo in un controllo TextBox

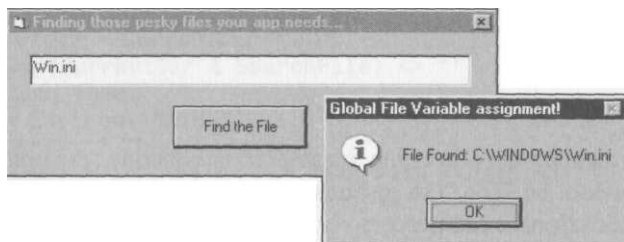
Per selezionare il testo in un controllo TextBox quando il controllo riceve il focus, come nell'esempio File Find, aggiungere il seguente codice all'evento GotFocus del controllo:

```
Private Sub txtFile_GotFocus()
   txtFile.SelStart = 0
   txtFile.SelLength = Len(txtFile.Text)
End Sub
```

Il motivo principale per scegliere questa soluzione è se si pensa che l'utente di norma desideri modificare il testo.

```
Private Sub RepairIt(FileIn As String)
    CommonDialog1.filename=FileIn
    CommonDialog1.InitDir =  App.Path
    CommonDialog1.ShowOpen
    txtFile.Text  = CommonDialog1.filename
    cmdFind_Click
End  Sub
```

Alla fine della procedura viene chiamato nuovamente cmdFind_Click. Ho aggiunto un pulsante *Cancel* alla finestra di messaggio cmdFind per evitare di creare un circolo senza fine se il file non viene trovato e l'utente non può modificare la situazione.

Figura 16.13 *La routine FindFile può essere utilizzata per trovare i file necessari per un 'applicazione.*

Ovviamente non è necessaria questa applicazione per scoprire dove si trova Win.ini (si veda la Figura 16.13). Se però il buon funzionamento di un'applicazione dipende dalla presenza di determinati file, aggiungendo una procedura simile a questa per trovare un file si può rendere il codice più sicuro. Se non altro, la si può utilizzare per assicurarsi che l'applicazione abbia accesso al proprio file della Guida in linea e per copiare, se necessario, questo file nella locazione appropriata.

Ricorsione

L'argomento di questo paragrafo, la ricorsione, è in un certo senso opposto all'argomento del capitolo, l'ottimizzazione del codice. Nonostante i metodi ricorsivi siano spesso un modo elegante di risolvere i problemi di programmazione e possano utilizzare poco codice e riflettere con chiarezza la natura sottostante dell'algoritmo coinvolto, l'esecuzione dei programmi di VB che utilizzano la ricorsione quasi certamente sarà più lenta rispetto ai programmi non ricorsivi. In questo capitolo è stata discussa l'ottimizzazione di diversi aspetti: la velocità, la velocità apparente, il consumo di memoria e così via. Si può pensare alla ricorsione come a un modo per ottimizzare la chiarezza degli algoritmi.

Una procedura ricorsiva è una procedura che chiama se stessa. Se si può formulare un problema in modo che ogni passaggio abbia una soluzione ovvia o gli stessi parametri formali con cui si è iniziato, si ha un buon candidato per una soluzione ricorsiva. Si ha ricorsione profonda *quando vi sono molte chiamate di procedure ricorsive nidificate, quanto più è profonda la ricorsione, tanto più è probabile che il programma esaurisca la memoria o generi errori dello stack.*

La ricorsione diretta avviene quando una procedura chiama se stessa, come mostrato nell'esempio che segue: la *ricorsione indiretta* avviene invece quando una procedura chiama un'altra procedura che a sua volta chiama la prima (naturalmente in questo processo vi possono essere passaggi intermedi). Un esempio piuttosto banale di ricorsione indiretta si ha nel programma di esempio nel paragrafo precedente, Find-File Vbp: in determinate condizioni (quando non viene trovato il file) cmdFind_Click chiama la sottoroutine Repairlt, che termina con una chiamata ricorsiva indiretta a cmdFind_Click.

La ricorsione è utilizzata spesso nei programmi di intelligenza artificiale.

In teoria, il codice di qualsiasi programma ricorsivo può essere modificato per farlo diventare iterativo, generalmente producendo codice più veloce, ma meno chiaro. Una strategia di sviluppo plausibile consiste nel trova una soluzione ricorsiva che funzioni e quindi ricodificarla in modo iterativo per migliorare le prestazioni.

In VB, quando una procedura chiama se stessa utilizzando la ricorsione diretta o indiretta, le informazioni della procedura chiamata in modo ricorsivo devono essere registrate come se si trattasse della chiamata a una nuova procedura. Di conseguenza, una procedura ricorsiva che è nidificata all'interno di 100 chiamate genera un sovraccarico pari a quello di 100 chiamate diverse (i linguaggi ottimizzati per la ricorsione, come Prolog, eliminano buona parte di questo sovraccarico).

Per ridurre il supForto dello stack se si progetta una procedura ricorsiva profonda, si devono utilizzare il meno possibile i parametri formali. Se possibile, le variabili locali alla procedura devono essere dichiarate come statiche per evitare di creare dichiarazioni di variabili locali ridondanti. In questo caso è necessario tuttavia fare attenzione agli effetti collaterali (indesiderati) delle dichiarazioni statiche.

Esempio: la successione di Fibonacci

La successione di Fibonacci prende il nome da un matematico del tredicesimo secolo, Leonardo Pisano, noto anche come Fibonacci. Questa successione di numeri si ottiene iniziando da O e da 1, addizionandoli, e quindi aggiungendo i due numeri precedenti per ottenere il numero successivo. Questa serie è un candidato naturale per la generazione ricorsiva; il numero *N* nella successione di Fibonacci equivale a *N- 1* più *N- 2.*

In natura si trovano molti fenomeni riconducibili a questa successione matematica, per esempio la disposizione a spirale dei petali in un fiore. Leonardo Pisano "scoprì" per primo questi famosi numeri lavorando sul numero di topi che ci si doveva aspettare iniziando da una coppia. Il Listato 16.8 contiene la funzione ricorsiva che genera la successione di Fibonacci:

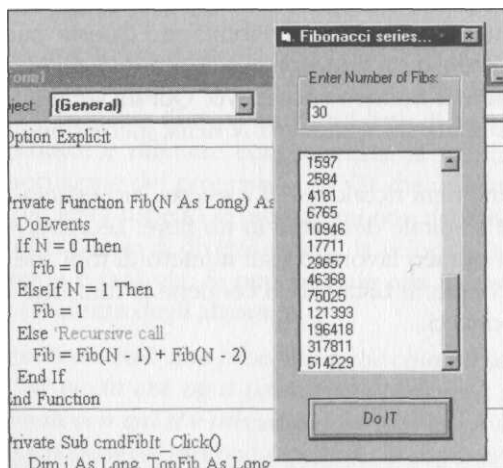
Listato } 6.8 *Calcolo della successione di Fibonacci.*

```
Private Function Fib(N As Long) As Long
   DoEvents
   If N = 0 Then
      Fib = 0
   ElseIf N = 1 Then
      Fib = 1
   Else 'Chiamata ricorsiva
      Fib = Fib(N - 1) + Fib(N - 2)
   End If
End Function
```

Il programma dimostrativo Fibonacci è salvato nel CD-ROM con il nome Fib. Vbp.

Con più di 25 numeri di Fibonacci in questo programma la ricorsione diventa molto profonda. Ho aggiunto una chiamata DoEvents alla funzione in modo che venga almeno ridisegnato lo schermo (e diventa più facile terminare l'operazione, se si decide di farlo). È possibile chiamare la funzione Fib e aggiungerla alla casella di riepilogo in un unico passaggio (si veda la Figura 16.14):

```
Private Sub cmdFibIt_Click()
   Dim i As Long, TopFib As Long
   Screen.MousePointer = vbHourglass
   lstFib.Clear
   TopFib = Val(txtHowMany)
   For i = 0 To TopFib
      IstFib.AddItem Str(Fib(i))  'Chiamata alla funzione Fib
   Next i
   Screen.MousePointer = vbDefault
End Sub
```

Figura 16.14 *È possibile utilizzare la chiamata a una funzione ricorsiva per generare la successione di Fibonacci.*

Esempio: il massimo comun divisore

L'esempio del massimo comune divisore, salvato nel CD-ROM con il nome Divisor.Vbp, utilizza una funzione ricorsiva per calcolare il massimo comun divisore di due numeri (la tecnica utilizzata per il calcolo va sotto il nome di algoritmo euclideo).

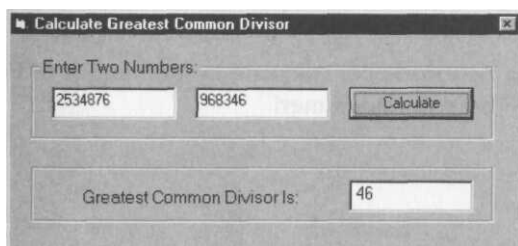
Il Listato 16.9 mostra la funzione ricorsiva:

Listato 16.9 *Calcolo del massimo comune divisore di due numeri.*

```
Private Function GreatestCommonDiv(N1 As Double, N2 As Double) _
   As Double
   If N2 = 0 Then
      GreatestCommonDiv = N1
   Else
      GreatestCommonDiv = GreatestCommonDiv(N2, N1 Mod N2)
   End If
End Function
```

Il seguente è il codice del modulo che chiama la funzione e che visualizza il risultato (come mostrato nella Figura 16.15):

```
Private Sub Command1_Click()
   Dim GCD As Double
   Label2 = ""
   GCD = GreatestCommonDiv(Val(Num1), Val(Num2))
   Label2 = Str(GCD)
End Sub
```

Figura 16.15 *È possibile utilizzare la ricorsione per calcolare il massimo comun divisore di due numeri.*

Riepilogo

In questo capitolo si è discusso di argomenti molto imFortanti relativamente all'ottimizzazione della velocità dei programmi. Inoltre si è parlato della velocità apparente: come tenere calmi gli utenti mentre il computer è impegnato per lunghi periodi nell'esecuzione di operazioni. Questi argomenti sono correlati alla creazione di programmi che fanno una buona impressione iniziale. Molti di essi mostrano inoltre come ci si può assicurare che il programma faccia esattamente questo.

In questo capitolo è stato anche spiegato come ottimizzare altri aspetti, oltre alla velocità. È stato mostrato come ottimizzare il codice per ridurre il consumo di memoria e sono state presentate numerose tecniche che permettono di ottimizzare altri aspetti: la compilazione condizionale, utile per ottimizzare il numero di piattaforme per le quali può essere compilato il codice, i file di risorse esterne, utili per ottimizzare il numero di lingue diverse che può visualizzare un programma e la ricorsione, utile per migliorare l'eleganza degli algoritmi.

- Sono stati forniti numerosi suggerimenti su come ottimizzare la velocità apparente, reale e di visualizzazione e il consumo di memoria di un programma.
- Si è appreso a utilizzare una schermata di avvio iniziale per nascondere le routine di inizializzazione.
- Si è scoperto come controllare l'avvio di applicazioni esterne tramite la funzione Shell.
- È stato spiegato come scegliere tra la compilazione in pseudocodice e la compilazione in codice nativo.
- Si è discusso dell'utilizzo dei commutatori del compilatore di codice nativo.
- Si è appreso a velocizzare il caricamento di applicazioni di grande dimensioni suddividendole e utilizzando la funzione Shell.
- È stato dimostrato come si utilizzano un file di risorse esterne e la compilazione condizionale per visualizzare versioni in lingue diverse di un'applicazione.
- Si è appreso come inserire codice per misurare la velocità di esecuzione.
- Si è scoperto come controllare l'esistenza e la posizione di file necessari alle applicazioni e come permettere all'utente di correggere eventuali problemi.
- Si è visto come usare la ricorsione per calcolare la successione di Fibonacci e il massimo comun divisore di due numeri.

SEGRETI
DI VISUALIZZAZIONE

PROGETTAZIONE DI UNA BUONA INTERFACCIA

- Come realizzare un'interfaccia adatta e soddisfacente per l'utente
- Come far divertire l'utente

Un corretto progetto di interfaccia nasce sulla carta quando lo sviluppo dell'intero software è ancora nelle sue fasi iniziali. Una buona interfaccia utente cammina di pari passo con un'architettura di codice ben strutturata; si completano a vicenda e lavoranoinsieme.
Sfortunatamente, non è il modo in cui normalmente si opera. Quello che di solito accade è che l'interfaccia compaia nel progetto del programma dopo che lo sviluppatore ha formalizzato i controlli e dopo averli realizzati nel codice, in altre parole, l'interfaccia è spesso un'aggiunta "posticcia".
In un mondo perfetto di progettazione software, il team di programmatori dovrebbe lavorare insieme a un team di designer di interfacce durante lo sviluppo del codice, con lo scopo di realizzare ciò che l'utente reputa bello da vedere, facile e anche divertente da usare. Sfortunatamente, questo non è un mondo perfetto. La maggior parte del software è implementata da programmatori che, nonostante possano essere eccellenti pensatori e sviluppatori, non sono abituati ad essere dei progettisti.
In questo capitolo verranno spiegati inizialmente i principi generali di progettazione che bisogna considerare nelle prime fasi di sviluppo. Per esempio, dovrete valutare le aspettative dell'utente, considerare sotto tutti i punti di vista l'interfaccia dell'applicazione e quale sarà l'approccio di chi la userà. Poi verranno affrontati argomenti più specifici di progettazione, fra cui il controllo delle azioni dell'utente e la gestione degli errori dell'applicazione in modo da non spaventare l'utente con inutili e complicati messaggi d'errore.

Il progetto di interfacce e il sedile posteriore

La maggior parte del software, attualmente in commercio, è come un'automobile con due persone a bordo: una che guida, l'altra che siede dietro. L'autista è il codice che permette all'applicazione di funzionare, ha il totale controllo dell'automobile e di solito non si prende cura della persona seduta dietro. Il passeggero è il progetto dell'interfaccia, seduto sul sedile posteriore con una cartina stradale, il quale cerca di suggerire una strada migliore che l'autista allegramente ignora.

Il software non è creato per il programmatore ma per chi lo usa. E sono gli obiettivi *e* le aspettative dell'utente che il programmatore deve ricordarsi quando progetta. Se gli utenti possono essere velocemente produttivi con un pacchetto software, lo compreranno e questo è denaro che andrà (si spera) nelle tasche dello sviluppatore.

"Va bene", potreste dire, "che cosa dovrei fare per compiacere gli utenti?". Non è facile dirlo. Ci sono quelli che non vogliono sapere cosa succede all'interno del computer, quelli che si muovono come elefanti nell'applicazione senza leggere né seguire le istruzioni *e* altri che si muovono come timidi gattini, letteralmente spaventati dal computer di per sé. (Non sto scherzando: ho visto persone preoccupate solo per accenderlo.) Con la tecnologia dei computer che continua a diffondersi, il numero di questi utenti, tecnologicamente ignoranti, continuerà ad aumentare.

Per questa ragione, anche nel mondo commerciale, molte persone sanno come usare un word processor e l'e-mail, ma niente più. I progettisti di programmi mirano, più dello sviluppatore, a capire il loro pubblico e a *realizzare* interfacce appropriate. Per esempio, Microsoft Word, è di per sé molto complicato da utilizzare. Generalmente gli utenti non sviluppatori hanno i loro obiettivi, più o meno esplicitamente dichiarati:

- "Non voglio sembrare stupido e intimidito".
- "Non voglio troppi fronzoli; voglio solo lavorare nel modo più veloce possibile".
- "Non voglio perdere niente di quello che ho prodotto nelle ultime nove ore".
- "Non voglio annoiarmi; voglio divertirmi".

Sfortunatamente, la maggior parte del software sul mercato fa tutto il possibile per confondere gli utenti, quasi aiutandoli a commettere errori, a sentirsi a disagio, a lavorare lentamente e a sentirsi annoiati e frustrati. Questo spesso *accade* perché gli sviluppatori di software sono così concentrati a pensare e implementare il codice che si dimenticano l'obiettivo per il quale hanno iniziato: creare un prodotto per l'utente.

Ci sono così tante interfacce mal *realizzate* nel mondo del software che risulta ingiusto scegliere degli esempi specifici. Senza dubbio potete velocemente fare una vostra lista dei "dieci peggiori". Ecco alcuni principi generali che dovete ricordare quando iniziate il design di un progetto software.

- Un buon progetto di interfaccia aiuta l'utente a ottenere risultati migliori. Siate puliti, coerenti e concisi nel design della vostra finestra come lo siete per l'architettura del vostro codice.

  Create un ambiente che funzioni come si aspetta l'utente. Per esempio se realizzate una finestra *Proprietà,* il pulsante *Applica* dovrebbe registrare i cambiamenti effettuati, ma non chiudere la finestra; invece, il tasto *Chiudi* deve sia registrare i cambiamenti che chiudere la finestra.
- Cercate di non nascondere in profondità le funzioni imFortanti; in altre parole, non fate che l'utente si perda messaggio dopo messaggio (o menu dopo menu) per trovare una certa proprietà o una data impostazione. Inoltre, dopo aver perso tempo nella ricerca, i poveri utenti dovranno eventualmente chiudere tutti questi messaggi o menu.
- Organizzate lo spazio della finestra raggruppando in maniera visiva gli argomenti correlati; sono di aiuto frame, riquadri e rettangoli. La maggior parte delle persone tende a guardare le cose da sinistra a destra, dall'alto verso il basso. Cercate di capire le abitudini delle persone per utilizzarle a vostro vantaggio.
- Siate creativi e fantasiosi con gli strumenti di design disponibili ma evitate di confondere l'utente con troppi colori o tipi di carattere. Questo approccio farà sembrare la vostra applicazione disorganizzata, e l'utente non saprà dove iniziare o sarà così frastornato da non capire neanche dove deve guardare. Ricordatevi il vecchio adagio: "meno è meglio".

Fino ad ora il progetto dell'interfaccia sembra essere stato ampiamente ignorato dalla comunità del software. Comunque, tutto ciò sta cambiando con l'avvento di Windows e la vasta espansione del mercato degli home computer. Un libro interessante con idee provocatorie è *About Face: The Essentials of User Interface Design* di Alan Cooper (IDG Books Worldwide).

Un'nterfaccia più amichevole

Il mondo dei computer continua a cambiare, evolvendosi in qualcosa di nuovo, differente, e migliore. L'interfaccia utente di dieci anni fa è ben diversa da a quelle odierne. Fortunatamente, oggi gli sviluppatori hanno più familiarità con il design di interfacce. Ora esistono più possibilità rispetto a prima. Prendete, per esempio, l'ambiente DOS: i suoi comandi, il suo schema bitonale di colori, e la sua incapacità di trattare più di una cosa alla volta. Ora, considerate la shell di Windows: la sua flessibilità, l'ambiente multitasking, l'innumerevole combinazione di colori, e il modo con cui un neofita può velocemente orientarsi ed essere produttivo. Windows fa lavorare il computer per l'utente, mentre il DOS rendeva difficoltoso il lavoro con il computer. Oggi, gli sviluppatori e i progettisti di interfacce sono impegnati a capire come integrare funzionalità Web in maniera utile e ad effetto con i loro programmi. Domani, chi lo sa?
Considerate questo paragone, quando state sviluppando un'interfaccia e la struttura di codice sottostante per un'applicazione, e cercate di capire cosa potete aggiun-

gere per renderla funzionale all'utente. La semplicità è sicuramente la chiave di lettura. Il vostro programma sarà più efficace e facile da usare se riuscirete a raggnippare gli elementi dell'interfaccia in insiemi di base logicamente correlati tra loro.

Partendo dalla premessa che semplice è meglio, è interessante notare come alcuni obiettivi siano piùfacili da realizzare in un ambiente a riga di comando come DOS o UNIX (almeno se sapete come fare).

Quando state progettando un interfaccia:

- Valutate il livello (o i livelli) di capacità dell'utente al quale è indirizzata. Dopo averlo determinato, spesso dovrete trovare un compromesso tra potenza e facilità d'uso.
- Pensate allo scopo del form e al numero minimo di controlli che potete aggiungere per ottenere facilità d'uso.
- Pianificate come il codice possa interagire con i controlli e viceversa. Per esempio, supponete di guarnire una coppa di gelato virtuale (come quella nel Capitolo 8). L'interfaccia potrebbe presentare una casella di controllo che l'utente selezionerebbe per dare l'ultimo tocco alla coppa. Sotto la casella di controllo potrebbe trovarsi una serie di pulsanti di scelta che l'utente utilizzerebbe per selezionare l'ultimo gusto di gelato da aggiungere in cima alla coppa. In una buona interfaccia, i pulsanti di scelta dovrebbero essere disabilitati fino a che l'utente non abbia selezionato la casella di controllo corrispondente.
- Pensate a ciò che l'utente non deve essere in grado di fare con il form, e restringete le sue possibilità di azione tramite il codice e le proprietà dei controlli.
- Non utilizzate troppe situazioni ripetitive, quali finestre di conferma. Per esempio, se è presente un pulsante *Aggiungi Cliente* che apre una finestra di dialogo *Aggiungi Cliente,* l'utente semplicemente farà clic su *Annulla* se non desidera aggiungere un cliente. Se l'utente volesse invece aggiungerlo, immetterebbe le informazioni e farebbe clic su OK. In molte applicazioni, invece, appare una casella di messaggio che dice: "Aggiunta di un cliente ai record" oppure altre sciocchezze di questo tipo. L'utente già sa che sta aggiungendo il cliente dal momento che ha fatto clic su OK; non sono necessarie conferme ulteriori che Forterebbero solo a dover chiudere un'altra finestra di messaggio. (Supponete che l'utente abbia 300 nuovi clienti: dovrebbe quindi fare clic sul pulsante OK nella finestra di conferma 300 volte!)

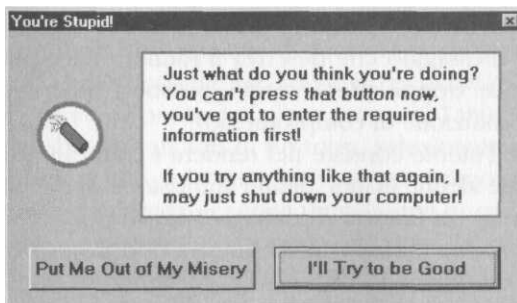
Dopo tutte queste raccomandazioni, non dimenticate che gli utenti vogliono solo divertirsi! Le persone imparano più velocemente e lavorano meglio se sono interessate a quello che stanno facendo. L'aggiunta di colori, caratteri speciali e caratteristiche su misura a un'interfaccia, può renderla più accattivante alla vista. Quando lo ritenete opFortuno, provate anche ad aggiungere qualcosa di stravagante che faccia sorridere l'utente.

Progettazione estetica di una interfaccia utente

Le tendenze di progetto per le interfacce utente sono molto variabili: alcuni metodi sono il risultato della variazione di imFortanti paradigmi di programmazione. Per esempio, la diffusione delle applicazioni SDI (Single Document Interface), rispetto alle applicazioni MDI (Multiple Document Interface) è dovuta alla capacità di poter lanciare più applicazioni sotto Windows. (Non è così imFortante essere capaci di aprire più copie di un "documento" in una applicazione, se è possibile ottenere lo stesso effetto aprendo più copie dell'applicazione). Un altro esempio può essere la crescita di popolarità di quelle applicazioni che hanno un'interfaccia molto simile a quella di un browser: questo è dovuto all'enorme successo che il Web ha avuto negli ultimi anni.
Insieme alle interfacce che imitano i browser, si è sviluppata la tendenza a inserire applicazioni all'interno del browser (sono i cosidetti *thin client* o clienti magari).
Ma alcune tendenze utilizzate per le interfacce utente risultano chiaramente inappropriate per una particolare applicazione (così come un singolo abito non veste bene su tutte le persone). Per esempio, alcune applicazioni non dovrebbero essere lanciate da una interfaccia di tipo browser. Applicazioni quali Photoshop e addirittura Visual Basic non sono appropriate per quel tipo di interfaccia.
È imFortante sapere distinguere tendenze valide, che sono il risultato del progresso tecnologico, dalle futili mode passeggere. Le vostre applicazioni appariranno moderne incorForando metodi validi, quando possibile, nelle vostre interfacce. Inoltre, utilizzando interfacce familiari agli utenti nelle vostre applicazioni, ne migliorerete la facilità d'uso.

Come controllare le azioni dell'utente in un ambiente guidato dagli eventi

Potreste dire "Aspetta un attimo, come è possibile controllare l'utente in un ambiente dove gli è possibile attivare qualsiasi controllo in ogni istante?" Così come nel caso di una folla allo stadio o a teatro, esistono modi per guidare l'utente di un'applicazione in un'opFortuna direzione senza che la cosa sia evidente e senza atterrirlo con la visualizzazione di una finestra di messaggio come quella mostrata nella Figura 17.1.

Figura 17.1
Dovreste guidare gli utenti nella giusta direzione, non spaventarli con messaggi come questo.

Se riuscite a pensare a un modo per guidare gli utenti, dite loro cosa fare, mettendo a disposizione, se necessario, un insieme limitato di opzioni. In questo modo, non dovranno costantemente cercar di capire che cosa devono fare, interrompendo il filo del ragionamento. Potete guidare l'utente in molti modi: disabilitando controlli, impostando valori predefiniti, stabilendo l'ordine nel quale si passa da un campo all'altro di un form.

Come rendere non disponibili le opzioni e assegnare valori predefiniti

Quando si crea un programma, dopo avere deciso quali controlli utilizzeranno e visualizzeranno le informazioni che il programma stesso gestirà, se ne dovrebbero fissare i valori di default. Questi valori suggeriscono all'utente quali tipi di valori o impostazioni si debbano assegnare a un particolare controllo. In una casella di testo contenente informazioni (per esempio nome e indirizzo di una persona), è possibile fissare tutti i valori di default. Per facilitare l'utente nell'immissione delle informazioni (così che non debba cancellare il testo di default per immettere il nuovo dato) assicuratevi che il testo immesso di default dal codice nelle varie caselle sia selezionato quando ci si sposta da una casella all'altra.

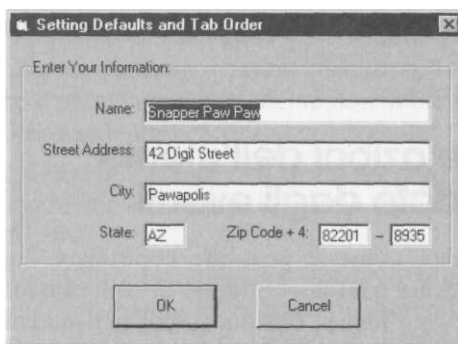
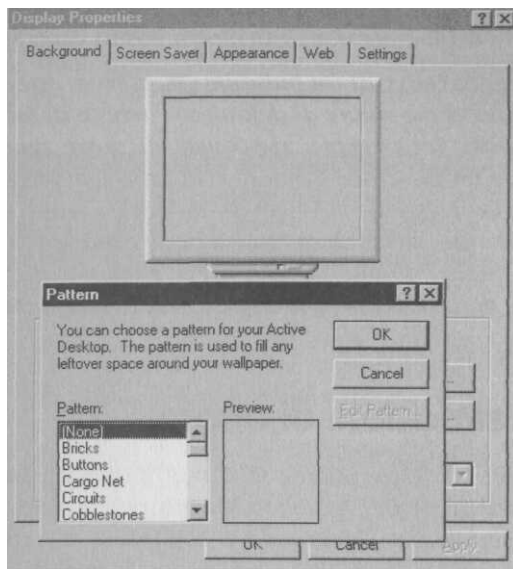
Figura]7.2 *La presenza di un testo predefinito nelle caselle di testo aiuta l'utente a capire che tipo di informazioni deve inserire.*

Nella finestra di dialogo mostrata in Figura 17.2 è anche possibile limitare il numero di caratteri inseribili dall'utente nei campi *State* e *Zip Code.* Se si tentasse di inserirne più di quanti ne accetta, il programma emetterà un segnale acustico, evidenziando il raggiunto limite di caratteri. Sarebbe piuttosto seccante fare visualizzare sempre al programma una casella di messaggio che descriva il numero massimo di caratteri inseribili (certamente, anche troppi effetti sonori possono innervosire l'utente, quindi si deve trovare una soluzione di compromesso).
Un altro modo per guidare l'utente consiste nel rendere i controlli non disponibili finché non si siano compiute alcune azioni, quali l'immissione di dati in una casella di testo o particolari selezioni. La finestra di dialogo *Pattern* (motivo) alla quale si accede dalla scheda *Background (Sfondo)* della finestra di dialogo *Display Properties (Proprietà - Schermo)* del Pannello di controllo di Windows visualizzata nella Figura 17.3 fa buon uso di questo tipo di approccio. L'utente non può fare clic sul pulsante *Edit Pattern (Modifica motivo)* prima di aver selezionato una struttura modificabile.

Figura 17.3 *La finestra di dialogo Pattern contiene un pulsante Edit Pattern che è disabilitato finché l'utente non seleziona una struttura.*

Tutte queste informazioni possono sembrarvi ovvie, ma date un occhiata al software presente sul mercato o che usate quotidianamente in fase di sviluppo. Molti programmi non tengono in alcun modo conto dei più semplici problemi dell'utente.

Come assegnare una sequenza preordinata alla pressione del tasto Tab

Una finestra di applicazione deve essere "facile da percorrere" come lo è una casa. Se si cammina in un'abitazione non "scorrevole", lo spostamento da una stanza all'altra risulta difficoltoso e irregolare, perché le stanze sono poste in maniera non ordinata. Muoversi attraverso la casa risulterebbe quindi scomodo e vi sentireste disorientati a causa della struttura anomala della costruzione. Un form scorrevole deve avere un cursore che si muove seguendo una progressione preordinata alla pressione del tasto *Tab.* Questa sequenza associata alle successive pressioni del tasto *Tab* viene implementata mediante la proprietà TabIndex del controllo.

Oltre a facilitare il movimento dell'utente all'interno del form, la sequenza associata alle pressioni del tasto *Tab* permette di decidere quale sarà il controllo attivo all'apertura del form. È anche possibile escludere interamente dalla sequenza di *Tab* alcuni controlli semplicemente ponendo la loro proprietà TabStop a False.

Per determinare l'ordine di Tab di un form, selezionate il controllo, con la relativa proprietà TabIndex, che volete sia attivato per primo: può essere una casella di testo, una casella combinata, o anche un pulsante di comando. Poi spostatevi sulla finestra *Properties* e ponete la proprietà TabIndex del controllo scelto a 0. Selezionate il successivo controllo che volete rendere attivo nell'ordine di Tab e cambiate la sua proprietà TabIndex ad uno, e proseguite sempre in questo modo.

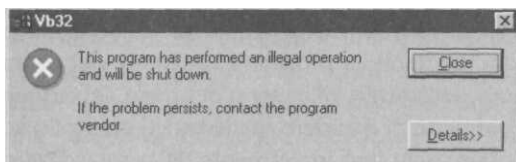
Dell'impostare le proprietà TabIndex dei controlli, assicuratevi di definire un ordine logico. Non ponete a caso l'ordine dei controlli. Muovetevi da sinistra a destra o

dall'alto verso il basso, per migliorare la scorrevolezza del form. Come già ricordato, fate in modo che le abitudini degli utilizzatori giochino a vostro vantaggio.

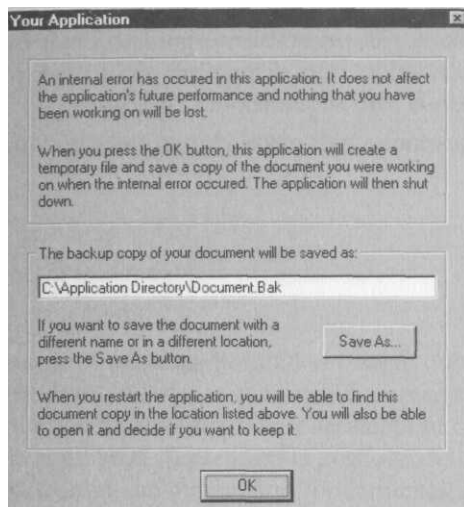
Anche se poneste la proprietà Default *di un pulsante* OK *a* True, *quest'ultimo non apparirebbe automaticamente come valore di default se l'arresto di tabulatore non fosse impostato correttamente. Per esempio, supponete di avere due pulsanti di comando, uno* OK *e l'altro* Cancel, *su un form dove la proprietà* Default *del pulsante* OK *sia posta a* True *e la proprietà* TabIndex *del pulsante* Cancel *sia posta a O. All'apertura del form, il pulsante* Cancel *avrà l'attivazione e quello* OK *non avrà il contorno scuro indicante quale è il bottone attivo. Per risolvere questo problema assicuratevi che nell'ordine di tabulazione per il pulsante* OK *venga prima del pulsante* Cancel.

Come gestire le situazioni di errore

È una verità ovvia affermare che ogni programma, non imForta quanto bene sia stato creato, si comForterà alcune volte in modo inappropriato: questa situazione viene definita come una *condizione di errore.* Le condizioni di errore possono essere molto limitate e influenzare la validità dei dati visualizzati dal programma, oppure essere tanto forti da provocare un crash di tutto il sistema. La causa che determina una condizione di errore può essere individuata in una programmazione non corretta, come per esempio l'accesso errato ad indirizzi di memoria, oppure causata dal sistema operativo o dall'hardware. L'unica cosa certa è che i vostri programmi, prima o poi, incontreranno una condizione di errore.
In realtà, il primo pensiero di ogni sviluppatore che si occupi della gestione degli errori presenti in un'interfaccia utente, dovrebbe essere: "Non voglio spaventare l'utente". Sicuramente voi avrete già incontrato la finestra di messaggio mostrata in Figura 17.4. Questa finestra può risultare fuorviante e minacciosa per gli utenti alle prime armi. (Un principiante di mia conoscenza, incontrando per la prima volta quella casella di messaggio, mi disse di non avere saputo più che fare e come procedere.)

Figura 17.4
// messaggio di errore che il programma visualizza non è per niente amichevole.

Ogni applicazione deve occuparsi degli errori e questi errori possono, certamente, essere intercettati. (Per i dettagli della gestione degli errori, si veda il Capitolo 15). Invece di visualizzare un terribile messaggio costituito da una grossa "X" rossa e da parole minacciose quali "illegai", potreste visualizzare una casella di messaggio per ogni particolare errore, la quale descriva accuratamente il problema e dica all'utente in un linguaggio semplice che cosa debba fare. O al limite potreste non visualizzare alcun messaggio per un particolare problema e potrebbe essere il programma stesso a farsene carico, gestendo l'errore in maniera trasparente all'utente. La Figura

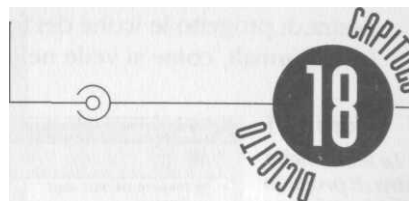
17 5 mostra una casella di messaggio migliore di quella presentata in Figura 17.4. Inoltre la casella di messaggio di Figura 17.5, non visualizzando all'utente informazioni confuse, quali quelle che si ottengono alla pressione del pulsante *Details», lo* rassicura e gli dice che cosa fare e quali siano le opzioni possibili.

Figura 17.5 *Questa finestra di messaggio fornisce le stesse informazioni di quella di Figura 17.4, ma senza incutere timore.*

Così, invece di aggiungere a una applicazione messaggi di errore, imprecici e terroristici quali "Hai sbagliato!" oppure "Procedura Illegale, chiusura del programma!", dovreste ridurre al minimo i messaggi di errore e visualizzarli solo nei casi in cui l'utente debba intraprendere azioni specifiche come il salvataggio di un file che altrimenti andrebbe perso.

Riepilogo

Progettare un'interfaccia utente interessante e concisa è difficile tanto quanto lo sviluppo del codice per l'applicazione. Purtroppo l'interfaccia viene abitualmente trascurata fino alla fine del progetto (o non considerata del tutto).
I programmi sono creati dagli sviluppatori esclusivamente per gli utenti. In molte situazioni questo punto imFortante viene dimenticato a causa dell'atmosfera agitata che generalmente accompagna lo sviluppo di un progetto. Se date un'occhiata al software disponibile sul mercato, scoprirete interfacce generalmente scomode per l'utente, che lo Fortano a sentirsi incapace di utilizzare il programma e lo rallentano nel lavoro.
Durante il normale evolversi di un progetto, alle interfacce utente viene data molta meno attenzione rispetto a ogni altro aspetto del programma. Alcune imFortanti applicazioni vengono persino scritte senza la consulenza di un progettista di professione. Paradossalmente, sembra proprio che il successo o il fallimento commerciale di molte applicazioni dipenda strettamente dalla loro interfaccia utente. Quando si crea un'applicazione e la sua interfaccia, è opFortuno seguire queste regole:

- Realizzare le aspettative dell'interfaccia utente e quindi avvicinarsi a ciò che l'utente desidera.
- Permettere all'utente di divertirsi e aiutarlo a velocizzare il lavoro.
- Procedere con metodologie di progetto che sfruttino miglioramenti tecnologici (ma non quelli effimeri o chiaramente insensati).
- Controllare le azioni dell'utente in un ambiente guidato dagli eventi, fissando valori di default, rendendo non disponibili alcuni controlli sotto certe condizioni e stabilendo la loro sequenza di attivazione.
- Non utilizzare metodi di gestione degli errori che spaventino l'utente e lo confondano.

APPLICAZIONI MDI E MENU

- Come creare e maneggiare progetti MDI
- Come gestire menu

La pianificazione di un progetto MDI *(multiple document interface)* è imFortante per poter permettere all'utente di aprire documenti dello stesso tipo contemFora-neamente. Questo capitolo descrive le modalità di creazione e gestione della struttura di applicazioni MDI. Tratteremo la gestione dei menu in questo stesso capitolo perché le funzionalità dei menu sono fondamentali per l'interfaccia MDI.

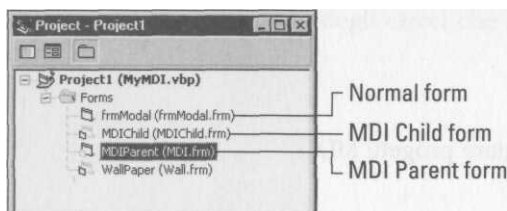
Come creare applicazioni MDI

Le applicazioni MDI prevedono più form *figli* in un unico form *genitore.* Invece, le applicazioni SDI, *single document interface,* si basano su un'unica finestra, utilizzata per tutte le interazioni con l'utente. Blocco note e WordPad sono applicazioni SDI; sono classici esempi di applicazioni MDI Word per Windows ed Excel. Le applicazioni MDI permettono all'utente di gestire, in modo semplice, differenti insiemi di dati nello stesso istante, per esempio trascinando informazioni tra finestre figlie e utilizzando un menu *Window* per spostarsi tra le diverse finestre.

VB Application Wizard permette una terza possibilità di progetto, oltre all'MDI e all'SDI: lo stile Explorer. Iprogetti realizzati con modalità Explorer sono applicazioni SDI dotate di un riquadro ad albero gerarchico nello stile di Microsoft Internet Explorer e Windows Explorer (Esplora risorse).

Con l'avvento delle versioni di Windows a 32 bit con multitasking vero, le applicazioni MDI sono un po' meno di moda di quanto non fossero in passato.
Un esempio di questo cambiamento di rotta è il passaggio da un File Manager come quello di Windows 3.x , applicazione totalmente MDI, all'Explorer (Gestione risorse/Esplora risorse) di Windows 95/98. È necessario aprire più finestre di Explorer per gestire operazioni che si sarebbero potute effettuare utilizzando più finestre secondarie di File Manager. Ciononostante, il progetto di applicazioni MDI è molto adatto in molte situazioni, in particolare quelle centrate sul documento o basate su più esemplari di un unico modulo finanziario.

In Visual Basic, un'applicazione MDI contiene uno e un solo forni genitore, denominato *forni MDI.* Si può aggiungere a un'applicazione il form MDI utilizzando il menu *Project* (o i menu di scelta rapida nel Project Explorer).
Durante l'esecuzione del programma i form figli sono all'interno del form genitore (sebbene ciò non sia vero nella fase di progettazione). Per creare un form figlio MDI, bisogna impostare a True la proprietà MDIChild di un form normale. Nella finestra di progetto le icone dei form MDI genitori e figli sono differenti da quelle dei form normali, come si vede nella Figura 18.1.

Figura 18.1 *Le icone nella nestra di progetto identificano form MDI genitori e figli e form normali.*

I form all'interno di un'applicazione MDI assumono, in fase di esecuzione, queste caratteristiche peculiari:

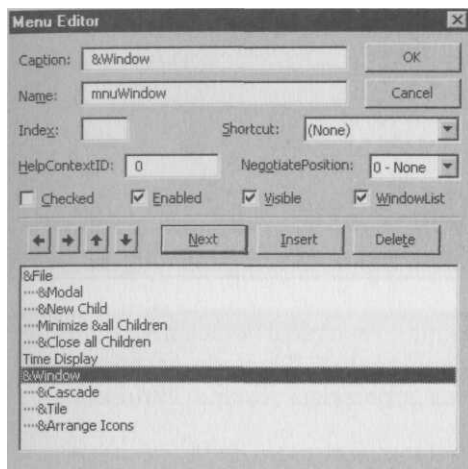
- Tutti i form figli sono all'interno dell'area *client* del form genitore MDI.
- La riduzione a icona di un form figlio appare nell'area del form MDI anziché sul desktop dell'utente. Minimizzando il form MDI non solo si minimizzano i form genitori ma anche i loro figli; il "gruppo di famiglia" è rappresentato da una sola icona. Il ripristino del form MDI comForta anche quello dei form figli allo stato precedente la riduzione ad icona.
- Il titolo di un form figlio ingrandito è riFortato concatenato a quello del form MDI nella barra di titolo della finestra.
- Di default, i menu del form figlio attivo, se esistono, sono sovrapposti a quelli del form genitore MDI (si veda il paragrafo "Conflitti tra menu" in questo capitolo). I menu non sono visualizzati sul form figlio attivo; in altre parole i form MDI figli fanno molte cose, ma non visualizzano dei menu all'interno dei propri bordi.

Gestione dei form figli

Per esemplificare una corretta gestione di un form MDI figlio, mostrerò in principio la struttura di un'applicazione MDI (presente sul CD-ROM con il nome di MyMDI. Vbp). Inizialmente la dimostrazione metterà in luce i problemi derivanti da quella che può sembrare la maniera più semplice di gestire i form figli.

Come primo passo, ho usato il Menu Editor per aggiungere un menu al form genitore. Questo fornisce un meccanismo per poter aggiungere form figli e visualizzare un *menu finestra (Window),* come mostrato nella Figura 18.2. Un menu finestra è un menu speciale che visualizza, al di sotto di una barra di separazione, i titoli di tutti i form figli aperti (come si può osservare nelle Figure 18.4 e 18.5). Di solito in

cima al menu finestra si inseriscono (ma non obbligatoriamente) voci per gestire i form figli quali *Cascade, Tile* e *Arrange Icons (Sovrapponi, Affianca* e *Disponi icone)* Per far si che un menu pricipale diventi di tipo finestra, basta semplicemente selezionare la voce corrispondente nel Menu Editor e controllare che sia attiva la voce *WindowList.*

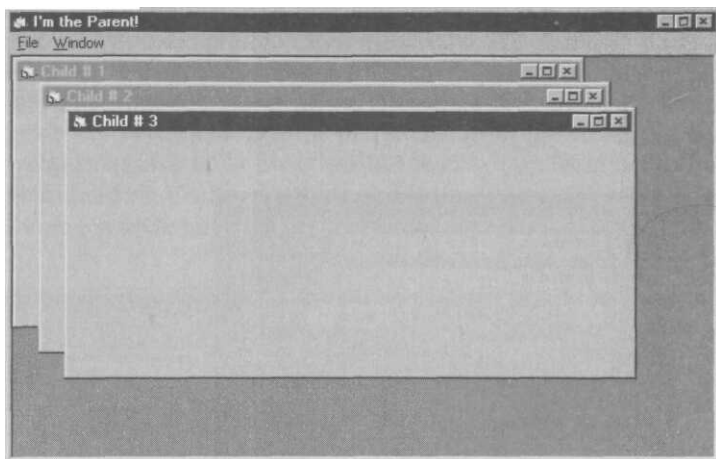
Figura 18.2
Per trasformare un menu principale in un menu finestra MDI, controllate che sia selezionata la voce WindowList.

Ho aggiunto una variabile pubblica, ChildCount, a MDIParent per poter tener traccia del numero di form figli aperti. ChildCount è impostata a O nell'evento load di MDIParent. Una nuova istanza di form MDI figlio è creata e mostrata con l'evento clic associato a mnNEW:

```
Option Explicit
Public ChildCount As Integer

Private Sub MDIForm_Load()
   ChildCount = 0
End Sub

Private Sub mnuNew_Click()
   Dim X As New MDIChild
   ChildCount = ChildCount + 1
   X.Show
End Sub
```

L'evento Load del form MDIChild è realizzato in maniera tale che ogni istanza del figlio mostri nel titolo la variabile ChildCount:

```
Private Sub Form_Load()
   Me.Caption = "Child # " + CStr(MDIParent.ChildCount)
End Sub
```

E possibile aggiungere nuovi form figli alla MDIParent, come si vede nella Figura 18.3. Se non si elimina mai un form figlio, chiudendolo, ChildCount manterrà traccia dei figli.

Figura 18.3 *Potete aggiungere quante istanze figlie desiderate la MDI genitrice.*

Cambiando argomento per un attimo, vediamo il codice necessario per rendere funzionale la parte superiore del menu finestra. Il metodo `Arrange` di `MDIParent` è invocato, come argomento, con una appropriata costante intrinseca:

```
Private Sub mnuCascade_Click()
   Me.Arrange vbCascade
End Sub

Private Sub mnuTile_Click()
   Me.Arrange vbTileHorizontal
End Sub
Private Sub mnuArrange_Click()
   Me.Arrange vbArrangeIcons
End Sub
```

La Figura 18.3 mostra form sovrapposti; la 18.4 mostra invece il risultato della "disposizione" delle icone (tutti i form figli ridotti a icona sono allineati nella parte inferiore di `MDIParent`); la Figura 18.5 mostra i figli affiancati orizzontalmente.

Guardando attentamente le Figure 18.4 e 18.5, vedrete il display di un orologio digitale sulla barra dei menu. Per aggiungere l'orologio, ho inserito un'opzione di menu chiamata `mnuTime` *sulla barra dei menu, nella posizione in cui volevo fosse visualizzato. Ho deselezionato la proprietà di visibilità di* `mnuTime`. *// titolo di mnuTime non è imFortante, perché non verrà mai visualizzato; comunque dovrete inserire qualcosa, altrimenti il Menu Editor non accetterebbe* `mnuTime`. *Ho impostato la proprietà di abilitazione del timer a* `True`, *e il suo intervallo a 1000 (vale a dire un secondo). Il codice dell'evento timer imposta* `mnuTime` *a* `Visible` *se non lo era già. e aggiunge un piccolo orologio nella sua posizione sulla barra dei menu:*

```
Private Sub Timer1_Timer()
   If Not mnuTime.Visible Then mnuTime.Visible = True
   mnuTime.Caption = Format(Now, "h:nn:ss AM/PM")
End Sub
```

Figura 18.4 *Iniziano a comparire delle lacune nei numeri assegnati ai form figli*

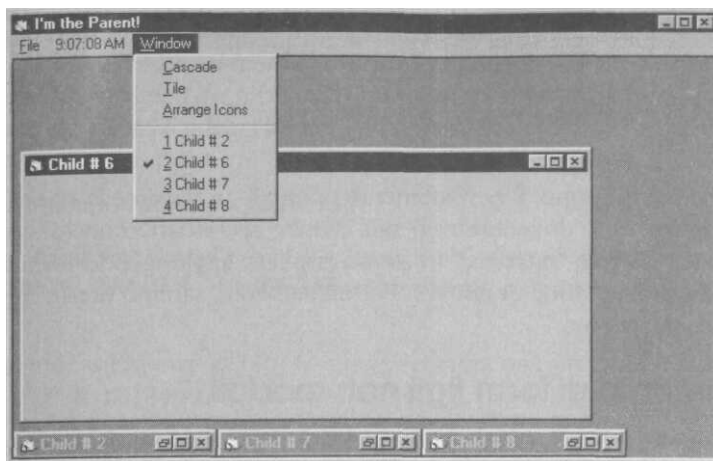
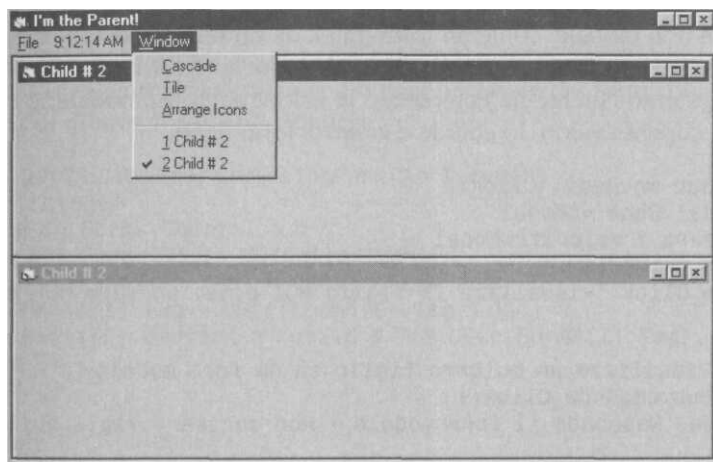



Figura 18.5 *Senza un adeguato sistema di conteggio, due form figli potrebbero avere lo stesso numero identificativo.*

Ho anche aggiunto un controllo RichTextBox *a* MDIChild. *Con il controllo originariamente posto sull'angolo in alto a sinistra dell'area client della* MDIChild, il *seguente codice, inserito nell'evento* Resize *della* MDIChild, *garantisce che il controllo* RichTextBox *si esspanda e si contragga per riempire l'intera area client del form, indipendentemente dalla sua dimensione:*

```
Private Sub Form_Resize()
    RichTextBox1.Height=Me.ScaleHeight
    RichTextBox1..width  =  Me.ScaleWidth
End Sub
```

Tornando alla struttura dell'MDI, è chiaro che ci sono problemi alla chiusura dei forni figli. Come si vede nella Figura 18.4, appaiono delle lacune nella successione dei numeri identificativi dei form.

Potete migliorare leggermente la situazione aggiungendo codice alla MDIChild che riduca ChildCount ogni volta che viene scaricata un'istanza:

```
Private Sub Form_Unload(Cancel As Integer)
   MDIParent.ChildCount = MDIParent.ChildCount - 1
End Sub
```

Non è ancora il massimo! È perfettamente possibile trovarsi nella situazione mostrata nella Figura 18.5, dove esistono più istanze secondarie con lo stesso numero identificativo. (Potete "ottenere" lo stesso risultato aggiungendo Form figli, cancellandoli, e aggiungendone di nuovi.). Fortunatamente, saremo presto in grado di gestire questa situazione!

Caricamento di form figli non modali

L'errore numero 401, "Can't show non-modal form when modal form is displayed", può sembrare sconcertante nel contesto di applicazioni MDI fino a quando non vi ricorderete che per definizione tutti i form figli sono non modali. Non potete caricare un form non modale, come un form figlio, da un form modale.Per evitare questo errore, inserite in una procedura il codice che apre sia il form modale sia quello non modale. Quando l'utente ha completato le azioni nel form modale, il codice lo nasconde, recupera i valori, lo chiude e carica il form figlio:

```
Private Sub mnuModal_Click()
   frmModal.Show vbModal
   'Recupera i valori frmModal
   Unload frmModal
   mnuNew_Click 'visualizza il figlio MDI p.es. un form non modale
End Sub

'Chiude/Visualizza un pulsnte figlio su un form modale
Private Sub cmdShow_Click()
   Me.Hide 'Nasconde il form modale - non caricare figli qui!
End Sub
```

Una struttura per tener traccia dei figli

Il problema, con lo schema di identificazione dei figli sviluppato fin qui, è che, quando viene cancellato un form figlio, agli altri form con numero identificativo superiore non viene assegnato un nuovo numero (diminuito di una unità). C'è una soluzione! Modifichiamo la procedura che aggiunge una nuova istanza figli in modo che memorizzi l'identificatore assegnato all'istanza nella proprietà Tag dell'identificatore:

```
Private Sub mnuNew_Click()
   Dim X As New MDIChild
   ChildCount = ChildCount + 1
   X.Tag = CStr(ChildCount)
   X.Show
End Sub
```

I titoli possono ora essere generati in ogni MDIChild *in base alla proprietà* Tag *dell'istanza. Notate però che il titolo deve essere impostato nell'evento bel ivate del* form *e non nell'evento* Load *delform.(Il riferimento a una proprietà del forni causa il caricamento delform. Il comando* X. Tag = *carica* X *ma il suo* Tag *non è ancora stato impostato.)*

Ecco il codice di Activate della MDIChild:

```
Private Sub Form_Activate()
   Me.Caption = "Child # " + Me.Tag
End Sub
```

Quando si chiude un'istanza MDIChild questa chiama una procedura pubblica che fa parte del modulo MDIParent, passando come argomento il valore del proprio tag:

```
Private Sub Form_Unload(Cancel As Integer)
   MDIParent.SetChildNumbers Val(Me.Tag)
End Sub
```

La procedura SetChildNumbers usa la collezione dei form per percorrere tutte le istanze MDIChild, aggiustando quelle con un tag maggiore del form che si sta chiudendo, prima di ridurre ChildCount di uno.

```
Public Sub SetChildNumbers(ClosingForm As Integer)
   Dim I As Integer
   For I = 0 To Forms.Count - 1
      If TypeOf Forms(I) Is MDIChild Then
         If Val(Forms(I).Tag) > ClosingForm Then
            Forms(I).Tag = CStr(Forms(I).Tag - 1)
            Forms(I).Caption="Child#"&CStr(Forms(I).Tag)
         End If
      End If
   Next I
   ChildCount = ChildCount - 1
End Sub
```

Un altro modo efficace per tener traccia delle istanze dei form figli è creare una matrice dinamica basata su una struttura definita dall'utente che contenga le informazioni suiform figli. Per esempio:

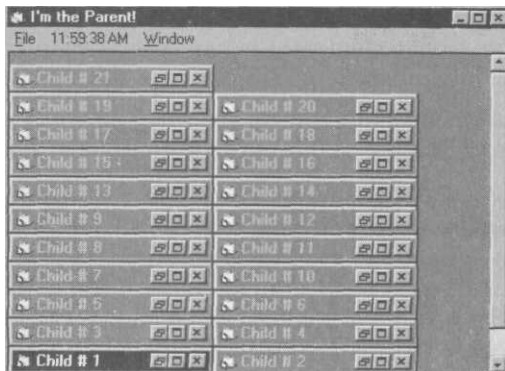
```
Type FormState
   FileName As String
   Dirty As Boolean
   Deleted As Boolean
End Type
```

Potreste essere obbligati a utilizzare questa tecnica quando dovrete archiviare informazioni specifiche (come il nome di un file) con ogni istanza di form figlio.

Riduzione a icona di tutti i form figli

Potete utilizzare la collezione dei form pr un altro scopo: minimizzare tutti i form MDIChild aperti (come nella Figura 18.6). Ecco il codice per poterlo fare:

```
Private Sub mnuMin_Click()
   Dim I As Integer
   For I = 0 To Forms.Count - 1
      If TypeOf Forms(I) Is MDIChild Then
         Forms(I).WindowState = vbMinimized
      End If
   Next I
End Sub
```

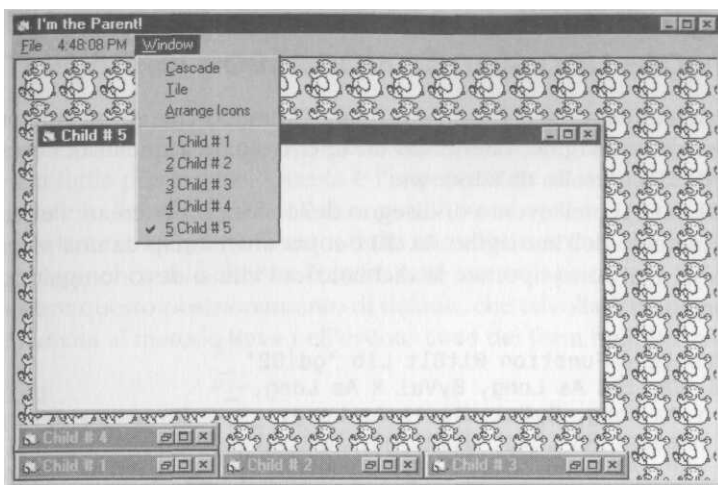
Figura 18.6 *Potete utilizzare la collezione dei form per eseguire un 'azione, per esempio la riduzione a icona, su tutti i form di un tipo particolare.*

Chiusura di tutti i form figli

Potete chiudere tutti i form MDIChild aperti in maniera simile, ma c'è un problema. Se liberate i form che vanno da O al valore della proprietà di conteggio della collezione meno uno, si presenterà un errore poiché la variabile che tiene traccia delle iterazioni non viene modificata quando il conteggio della collezione diventa più piccolo a seguito della chiusura di qualche form. Potete risolvere il problema procedendo nella chiusura dal form con contatore più alto fino a quello con valore 0:

```
Private Sub mnuClose_Click()
   Dim I As Integer
   For I = Forms.Count - 1 To 0 Step -1
      If TypeOf Forms(I) Is MDIChild Then
         Unload Forms(I)
      End If
   Next I
End Sub
```

Come creare sfondi per una applicazione MDI

Potete facilmente impostare uno sfondo per un'applicazione MDI, diverso da quello monocolore di clefault del form MDI genitore, come si può vedere nella Figura 18.7. Il trucco consiste nell'aggiungere un form figlio che contenga lo sfondo. Nel programma di esempio, il form si chiama Wallpaper. (Perché Wallpaper sia un form figlio bisogna impostare la sua proprietà MDIChild a True.)

Figura 18.7 *È facile aggiungere un effetto "carta da parati" di sfondo a una applicazione MDI.*

Potete caricare qualsiasi immagine (per esempio un file .Bmp o .Wmf) in Wallpaper impostandone la proprietà Picture. Poi aggiungete codice all'evento Activate di Wallpaper, che manipola lo ZOrder *(ZOrder è* la proprietà che gestisce l'ordine in cui sono disposte le finestre) del form, così che stia dietro agli altri forni figli:

```
Private Sub Form_Activate()
   Me.ZOrder 1
End Sub
```

Ora aggiungiamo codice a Wallpaper e a MDIParent (con il metodo Move) per essere certi che la dimensione della genitrice MDI e quella del suo sfondo siano sempre coerenti anche in caso di ridimensionamento:

```
' in Wallpaper
Private Sub Form_Resize()
   If Me.WindowState = vbNormal Then
      Me.Move 0, 0, MDIParent.ScaleWidth, _
         MDIParent.ScaleHeight
   End If
End Sub

' in MDI Parent
Private Sub MDIForm_Resize()
   WallPaper.Move 0, 0, Me.ScaleWidth, _
      Me.ScaleHeight
End Sub
```

Questo è il trucco. Per essere sicuri che i form figli ridotti a icona restino in primo piano (Figura 18.7), si aggiunge una riga di codice all'evento Resize di MDIChild:

```
Private Sub Form_Resize()

   !f WindowState = vbMinimized Then ZOrder 0
End Sub
```

Impiego di BitBlt per creare uno sfondo ripetitivo

Se volete potete impostare il form Wallpaper in modo che venga ripetuta al suo interno un stessa immagine, ottenendo un effetto simile a quello del comando *Tiled* del Pannello di controllo di Windows.
Si usa l'API BitBIt nell'evento di disegno dello sfondo per creare l'effetto di affiancamento ripetitivo dell'immagine. BitBIt copia una bitmap da una sorgente ad una destinazione. Qui sono riFortate le dichiarazioni che si devono aggiungere al form Wallpaper:

```
Private Declare Function BitBlt Lib "gdi32" _
    (ByVal hDestDC As Long, ByVal X As Long, _
    ByVal Y As Long, ByVal nWidth As Long, _
    ByVal nHeight As Long, ByVal hSrcDC As Long, _
    ByVal xSrc As Long, ByVal ySrc As Long, _
    ByVal dwRop As Long) As Long
Const SRCCOPY = &HCC0020
```

Dovrete aggiungere una bitmap come immagine di sfondo. Deve essere almeno di 32 x *32pixel. (Per veri/icario potete utilizzare la bitmap da me fornita, Bear.Bmp, o una delle tante installate da Windows nel sistema.)*

Controllate che la proprietà AutoRedraw *di* Wallpaper *sia posta a* False. *Se fosse posta a* True, *allora Visual Basic aggiornerebbe automaticamente la visualizzazione del form e ignorerebbe l'evento* Paint. *Quando* AutoRedraw *è impostata a* False *il form viene ridisegnato con il codice presente nell'evento* Paint.

Il Listato 18.1 contiene il codice che affianca le bitmap. Dovete posizionarlo nell'evento Paint di Wallpaper:

Listato 18.1 *Come creare uno sfondo ripetitivo.*

```
Private Sub Form_Paint()
   Dim X As Integer, Y As Integer, D As Long
   Dim PatternHeight As Integer, PatternWidth As Integer
   Dim SM As Integer
   ScaleMode = vbPixels
   PatternHeight = 32
   PatternWidth = 32
   For X = 0 To ScaleWidth Step PatternWidth
      For Y = 0 To ScaleHeight Step PatternHeight
         D = BitBlt(hDC, X, Y, PatternWidth, _
            PatternHeight, hDC, 0, 0, SRCCOPY)
      Next Y
   Next X
   ScaleMode = SM 'probabilmente twip come unità di misura
```

Come modificare la posizione dei form figli nell'evento Load

Quando visualizzate un nuovo form MDIChild, esso viene automaticamente dimensionato e posizionato nella successiva posizione in cascata, vale a dire a destra e sotto la finestra figlia precedente. Questa è l'impostazione di default anche se vengono chiuse tutte le finestre figlie. (In altre parole, se si aprono tre finestre figlie e poi le si chiude tutte, una quarta finestra figlia verrebbe aperta nella posizione che avrebbe occupato nel caso le altre tre fossero state ancora aperte.)
Potete modificare questo posizionamento di default, che talvolta è inadeguato, inserendo una chiamata al metodo Move nell'evento Load dei form figli. Per esempio:

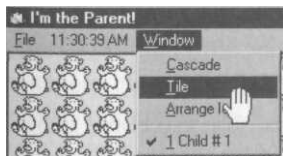
```
' in  MDIChild
Private Sub Form_Load()
   Move MDIParent.Width \ 4, MDIParent.Height \ 4,
      MDIParent.Width \ 2, MDIParent.Height \ 2
End Sub
```

Potete modificare questo codice per adattare la posizione di nuovi form figli a quella di altri, o comunque in maniera tale da soddisfare le specifiche esigenze della vostra applicazione.

Impostazione di un cursore personalizzato

Se volete, potete creare cursori personalizzati, chiamati anche *puntatori del mouse,* nella maggior parte delle applicazioni in grado di creare e modificare le risorse, come Visual C++. Il progetto di un cursore comprende un punto sensibile *(.hot spot)* che rappresenta l'area di attivazione del cursore (per esempio il punto sensibile nella freccetta di default è proprio la sua punta). Un cursore personalizzato può essere caricato nella proprietà MouseIcon di un form (e di molti altri oggetti) in fase di progettazione nella finestra *Properties.*

Potete impostare un cursore personalizzato per la vostra applicazione MDI. Un esempio è la mano, mostrata come cursore nella Figura 18.8. (Il file del cursore mano Paw.Cursi trova nella directory del progetto MyMDY nel CD-ROM allegato.)

Figura 18.8 *Un cursore personalizzato caricato nella proprietà MouseIcon.*

La proprietà MouseIcon *di un controllo* Label *è un buon posto per memorizzare cursori personalizzati perché* Label *aggiuntivi non comFortano un consumo elevato di risorse.*

Il cursore personalizzato viene poi attivato quando la proprietà MousePointer è impostata a vbCustom (uguale a 99). Per esempio, se all'interno della proprietà MouseIcon di una MDIParent è stato caricatoun cursore personalizzato, è possibile alternare l'impiego di cursori personalizzati e predefiniti:

```
MDIParent.MousePointer = vbCustom ' passa a cursore personalizzato(99)

MDIParent.MousePointer = vbHourGlass ' passa alla clessidra (11)

MDIParent.MousePointer = vbDefault ' (0)
```

Comunque, incontrerete una limitazione nell'impostare un cursore in questo modo: il cursore personalizzato appare solo nelle aree client del form MDI genitore e di quelli figli. In altre parole il cursore ritorna quello predefinito quando viene spostato sopra l'area della barra di menu dei form.

Per mantenere il cursore personalizzato ovunque nell'applicazione MDI, potete utilizzare le proprietà MouseIcon e MousePointer dell'oggetto schermo *(Screeri)*. Poiché non esiste modo di cambiare le proprietà dell'oggetto *Screen* in fase di progetto, il cursore personalizzato deve essere caricato nel codice. Ecco come potete realizzarlo nell'evento Load della MDIParent:

```
Private Sub MDIForm_Load()
    On Error Resume Next

    Screen.MouseIcon = LoadPicture("Paw.Cur")
    Screen.MousePointer = vbCustom
End Sub
```

Per inciso, quando aggiungete un cursore personalizzato con l'istruzione LoadPicture *come appena descritto, dovete Forre attenzione alla posizione relativa del file del cursore quando eseguite il progetto in modalità progettazione. Una possibilità è dare in forma esplicita nel codice il percorso completo del file. Una migliore alternativa è quella di Forre il file nella directory indicata dalla variabilie* Path *o nella directory attiva quando il progetto verrà attivato. Comunque notate che le directory attive variano in base alla modalità di attivazione del progetto. Se lo fate partire con un doppio clic sul file . Vbp, la directory contenente il. Vbp sarà quella attiva (e deve contenere il file del cursore). Se invece, fate prima partire il Visual Basic e usate il menu* File *per aprire il .Vbp, quella attiva sarà la directory di default di Visual Basic (e deve contenere il file del cursore).*

Se immmettete il nome del file del cursore senza percorso come nell'esempio

```
Screen.MouseIcon = LoadPicture("Paw.Cur")
```

dovete essere certi che il file sia ben posizionato, altrimenti il cursore modificato non verrà caricato. Inoltre, è una buona idea inserire un comando per gestire una condizione d'errore, quale On Error Resume Next, nella procedura di caricamento del cursore. Così, nel caso il cursore non venga caricato, il progetto verrà eseguito comunque senza visualizzare un messaggio d'errore .

Potete facilmente aggiungere una barra degli strumenti, o toolbar, a un'applicazione MDI utilizzando qualsiasi controllo contenitore che supForti una proprietà Align- Per esempio potete usare un controllo Picture. *// controllo* Toolbar *di Windows illustrato nel Capitolo 8, permette di aggiungere in modofacile e veloce barre deglistrumenti, come quelle di Windows, che supFortino automaticamente i ToolTip o suggerimenti (informazioni visualizzate quando il mouse passa sopra un controllo pulsante della barra degli strumenti). Il controllo CoolBar, sempre trattato nel Capitolo 8, permette di aggiungere barre degli strumenti eleganti con l'aspetto classico di Internet Explorer.*

Gestione dei menu

Il menu è una parte cruciale di ogni applicazione. Visual Basic fornisce il Menu Editor (Figura 18.2), uno strumento molto potente per la creazione di menu nella fase di progettazione, descritto nel Capitolo 2 e utilizzato in molti esempi di questo libro. In questo paragrafo ci concentreremo sulla manipolazione dei menu in fase di esecuzione, ovvero su come manipolare i menu nel codice.

Contese tra menu

Come probabilmente sapete, se un form figlio in una applicazione MDI contiene un menu, nel momento in cui viene attivato si impossessa della barra dei menu del form genitore, all'interno dell'applicazione MDI. Questo è vero anche se un oggetto OLE ha già preso possesso, a sua volta, della barra dei menu del form figlio. Come esempio, ritornando al progetto MyMDI.Vbp per un attimo, rimuovete il controllo RichTextBox dal form MDIChild e aggiungete un controllo contenitore OLE. Aggiungete al contenitore un nuovo oggetto, per esempio un documento di Word per Windows. (Per maggiori informazioni sull'uso dei controlli OLE, potete vedere il Capitolo 21.)

Aggiungete una voce di menu alla MDIChild, per esempio una singola voce intitolata *Hello* e chiamata mnuHello. Quando il form figlio con l'oggetto OLE inserito diventa attivo, il suo menu subentra a quello del genitore. E quando l'oggetto OLE, in questo caso Word per Windows, viene attivato, il suo menu prende possesso della barra dei menu di MDIChild che a sua volta fa la medesima cosa con la barra dei menu della MDIParent.

Supponete di volere che l'oggetto OLE incorForato abbandoni il menu della MDIChild (e successivamente anche il menu di MDIParent). Potete facilmente ottenere questo risultato ponendo la proprietà NegotiateMenus della MDIChild a False. Forre questa proprietà al valore False è come mandare all'oggetto OLE un messaggio del tipo "Non vorrai venire qui a comandare, torna a casa!" (Il valore di default di questa proprietà, da decidersi in fase di progettazione, è True.)

È anche possibile integrare menu di MDIChild con il menu dell'oggetto OLE, come mostrato nella Figura 18.9. Dopo aver posto al valore True la proprietà NegotiateMenus della MDIChild, potete utizzare la proprietà NegotiatePosition di una voce

del menu di livello più alto nella fase di progettazione per integrare la voce di menu con il menu dell'oggetto OLE. La proprietà NegotiatePosition può essere configurata in quattro modi diversi, come mostrato nella Tabella 18.1.

Figura 18.9 *La proprietà NegotiatePosition del menu Hello è stata posta a 2-Middle, posizionando Hello nel centro del normale menu OLE di Word per Windows.*

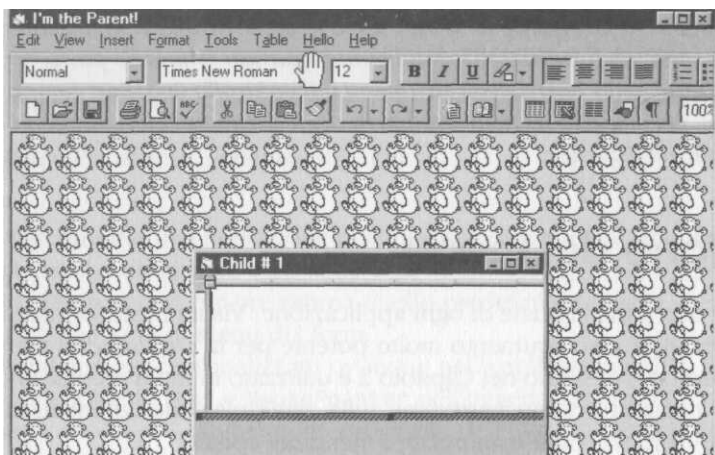



Tabella 18.1 *Valori della proprietà NegotiatePosition di una voce del menu di livello più alto.*

Valore	Conseguenza
0 - None	Questo è il valore predefinito per la proprietà. La voce del menu non viene visualizzata sulla barra dei menu quando l'oggetto OLE è attivo.
1 - Left	Il menu è visualizzato alla sinistra della barra dei menu quando l'oggetto OLE è attivo (a sinistra del menu dell'oggetto).
2 - Middle	Il menu è visualizzato da qualche parte nel centro della barra dei menu quando l'oggetto OLE è attivo, come si può vedere nella Figura 18.9
3 - Right	Il menu è visualizzato all'estrema destra della barra dei menu quando l'oggetto OLE è attivo (a destra del menu dell'oggetto).

Nell'esempio mostrato nella Figura 18.9, ho posto la proprietà NegotiatePosition del mnuHello a 2-Middle, e come risultato si è ottenuto che *Hello* venisse posizionato tra *Table* e *Help* nel normale menu *OLE* di Word.

Attribuzione di nomi ai menu

Il nome di un menu può essere considerato da due diversi punti di vista: quello dell'utlizzzatore e quello dello sviluppatore. Il nome che interessa all'utilizzatole è quello contenuto nella proprietà caption della voce di menu, e non il nome stesso della voce, che invece è quello che interessa principalmente agli sviluppatori.

Didascalia (Caption)

Le didascalie *(.Caption),* cioè le espressioni che comaiono nei menu e identificano le varie voci, devono rendere chiaro all'utente gli intenti delle diverse voci. In linea generale debbono essere univoche (anche se voci legate a differenti menu di livello superiore possano avere lo stesso nome).

Ogni voce di menu deve avere un tasto di scelta rapida (univoco per il suo livello). I tasti di scelta rapida permettono di accedere da tastiera agli eventi clic di un menu mediante pressione del tasto Alt insieme al tasto di una lettera. Potete assegnare un tasto di scelta rapida ponendo all'interno della voce del menu una "e commerciale" *(&)* di fronte al carattere prescelto. Di norma, si sceglie come tasto di scelta rapida la prima lettera della voce, a meno che non vi siano altre lettere che offrono una associazione mnemonica migliore (o che la prima lettera sia già usata)

Terminologia dei menu

Un menu è un oggetto menu top-level la cui intestazione (caption) compare sulla barra dei menu. Confusamente, la parola *menu è* anche usata per indicare, singolarmente o nel loro complesso, le voci contenute nella struttura di un menu. Il termine voce di menu si usa nella stessa maniera generale e un po' vaga.

Parlando in maniera rigorosa, una voce di menu *(menu item)* è un menu sottostante a un menu di livello superiore. Un *sottomenu è* invece una voce di menu con voci al suo interno, le quali sono chiamate *voci di sottomenu.* Una matrice di controllo menu è formata da un insieme di voci di menu sullo stesso menu (o sottomenu) le quali condividono lo stesso nome e le stesse procedure, con differenti menu nella struttura, distinguibili per il valore fisso di indice assegnato loro. Le Figure 18.10 e 18.11 mostrano come si combinino quelle parti in una struttura di menu.

Figura 18.10
Si usa il Menu Editar per aggiungere menu, voci di menu, sottomenu, voci di sottomenu e matrici di controllo menu alle finestre.

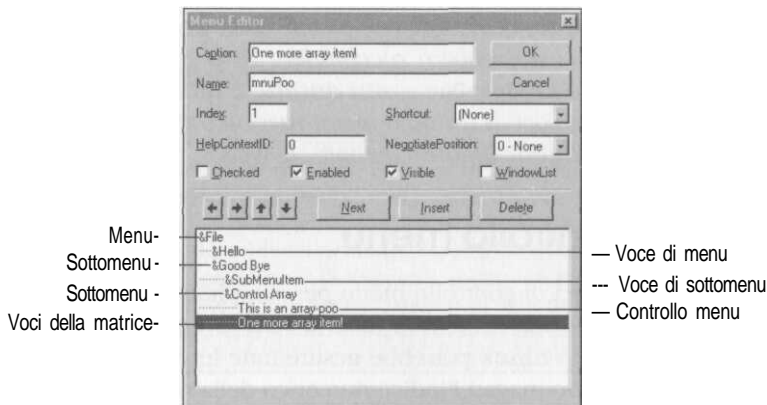
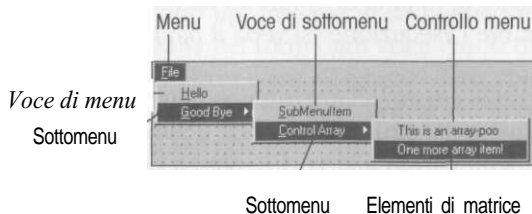



Figura 18.11
I menu possono contenere voci di menu, sottomenu e matrici di controllo menu.

Nomi interni dei menu

I nomi interni dei menu, ovvero il valore della proprietà Name dei menu, sono quegli identificatori che userete internamente per riferirvi al menu nel codice. Alcune culture "primitive" credono che, se si conosce il nome di qualcosa, se ne possiede il controllo totale. Ciò è certamente vero per quello che riguarda le voci di menu! Se non riuscite a trovarle, di certo non potrete utilizzarle per fare alcunché. Nel caso di progetti molto estesi, diventa particolarmente difficile *localizzare* le voci di menu nella gerarchla presente all'interno del Code Editor o nella finestra *Properties.*
La Microsoft suggerisce di far precedere al nome del menu il prefisso "mnu", indicando le voci di menu e sottomenu con la parola "item" e la matrice di controllo di menu con la parola "array", come mostrato nella Tabella 18.2.

Tabella 18.2 *Convenzioni suggerite dalla Microsoft per l'identificazione di menu all'interno del codice*

Elemento	Esempio
Menu	mnuFile
Voce di menu	mnuFileHelloItem
Struttura di menu	mnuFileGoodControlArray
Voce di sottomenu	mnuFileGoodSubltem

Il prefisso "mnu" *è* sicuramente utile, perché permette all'oggetto di essere subito riconosciuto come un menu; questi prefissi sono comunque opzionali.

Esisteperò qualcosa dipiù importante deiprefissi: scegliere nomi adeguatiper le voci di menu in modo che indichino chiaramente il loro contenuto. Utilizzando questa tecnica, le voci nel menu File *dovrebbero includere "File" come parte del loro nome, per esempio* mnuFileOpen, mnuFileSave, *e* mnuFileExit. *E se "File Open" avesse dei sottomenu, le loro voci dovrebbero racchiudere l'intera genealogia del contenuto:* mnuFileOpenRtf, mnuFileOpenDoc *e altri ancora. E tutto ciò darebbe anche il vantaggio di avere tutte le voci di un dato menu raggnippate, poiché il Code Editor e la finestra* Properties *ordinano alfabeticamente gli oggetti.*

Matrici di controllo menu

Potete utilizzare matrici di controllo menu per semplificare il codice quando potete usare un blocco di codice comune a tutte le voci in una matrice. Per esempio, lo stesso codice di evento click potrebbe gestire tutte le voci di menu in una matrice di controllo, utilizzando magari l'indice numerico della matrice per controllare l'esecuzione. È possibile utilizzare le matrici di controllo menu anche per creare voci di menu dinamicamente durante l'esecuzione (si veda il paragrafo "Gestione dinamica del menu" che segue).
Le matrici di controllo menu si creano all'interno del Menu Editor. Le voci della matrice devono essere contigue nella struttura di menu e allo stesso livello. Realizzate una matrice di controllo menu dando alle voci lo stesso nome ed assegnando loro un identificatore numerico progressivo (partendo da zero), come mostrato nella Fi-

gura 18.10. Il codice può essere facilmente assegnato all'evento Click della matrice di controllo menu a due elementi creata nella Figura 18.10 e mostrata nella Figura 18.11. Per esempio, per visualizzare una finestra di messaggio con il nome della voce della matrice di controllo menu su cui è stato fatto clic, bisognerebbe aggiungere il seguente codice all'evento:

```
private Sub mnuFoo_Click(Index As Integer)
   MsgBox  mnuFoo(Index).Caption
End Sub
```

Questo determina con precisione quale sia l'elemento della matrice che è stato attivato Ovviamente, potreste usare il valore indice per elaborazioni più raffinate. In altre parole, potreste utilizzare il valore indice in un gestore di evento Click della matrice di controllo menu al fine di determinare quale elemento della struttura sia stato scelto con un clic.

Menu pop-up

I *menu pop-up,* chiamati anche *menu di contesto* o (il nome ufficiale) *menu di scelta rapida,* sono strutture visualizzate in posizioni variabili quando vengono cliccati il tasto destro o quello sinistro del mouse. Per creare un menu pop-up, dovete utlizzare il Menu Editor per generare una serie di voci di sottomenu. Il menu genitore delle voci di sottomenu deve avere la proprietà Visible posta a False.

La Figura 18.12 mostra due menu pop-up e le loro voci, con la proprietà Visible *del menu principale non attivata. (Il progetto che mostra come invocare i menu pop-up è presente sul CD-ROM allegato con il nome PopDemo. Vbp.)*

Annidamento di sottomenu

Ogni menu di livello superiore in una finestra può avere fino a 5 livelli di sottomenu annidati. Ma solo perché è lecito, non è detto che lo si debba fare! Non create labirinti di sottomenu che renderebbero impossibile all'utente trovare la voce desiderata. Organizzate i menu in maniera intelligente. Cercate di rendere visibile ogni possibile voce di menu che il particolare contesto richiede. È possibile ottenere questo risultato abilitando e disabilitando l'accesso a determinati menu in particolari situazioni. Quando lo ritenete opFortuno, potete anche andare oltre e nascondere menu che non siano attinenti a un particolare lavoro o per un particolare utente.

Una volta creato il menu pop-up, insieme al suo genitore, nel Menu Editor, viene invocato il metodo PopupMenu del form ogni volta che si deve visualizzare il nuovo menu. Se non è specificato alcun form all'interno della chiamata al metodo PopupMenu, quest ultimo considera come valido il form attivo in quel momento. La sintassi per il metodo PopupMenu è la seguente

form.PopupMenu MenuName, Flags, X, Y, BoldCommand

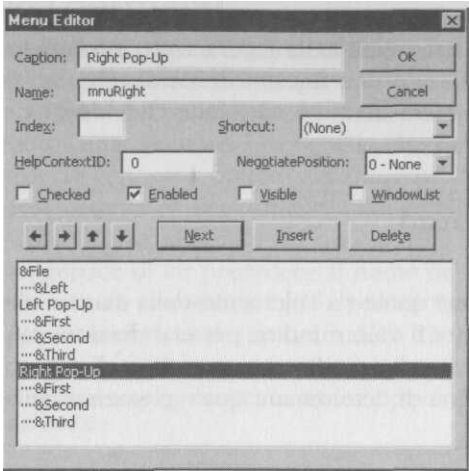
Figura 18.12
Per organizzare un menupop-up, bisognaperprima cosa porre laproprietà Visible del menu genitore a False.

Tutti i parametri sono opzionali eccetto il nome del menu, che è il nome del genitore invisibile delle voci del sottomenu pop-up. Il parametro *Flags* specifica i valori costanti selezionabili fra quelli nelle Tabelle 18.3 e 18.4. L'impostazione del parametro *Flags* controlla la posizione ed il comFortamento del menu. *X eY* rappresentano le coordinate alle quali il menu pop-up verrà visualizzato. Se le coordinate vengono omesse, il menu viene visualizzato nella posizione in cui si trovava il puntatore del mouse al momento del clic. Il parametro *BoldCommand* specifica quale nome di voce del sottomenu pop-up sia da visualizzare in grassetto all'interno del menu pop-up.
Sono disponibili due tipi di valori da immettere nel parametro *Flags: Location,* mostrato nella Tabella 18.3, e *Behavior,* nella Tabella 18.4. Potete utilizzare un valore per ognuno dei due tipi, unendoli con l'operatore OR.

Tabella 18.3 *Valori delparametro Flagsper il menupop-up (tipo Location).*

Nome costante	Valore	Descrizione
vbPopupMenuLeftAlign	0	La parte sinistra del menu pop-up *è* posta alla posizione X del metodo PopupMenu. Questo è il valore predefinito.
VbPopupMenuCenterAlign	4	Il menu pop-up è centrato alla posizione X.
vbPopupMenuRightAlign	8	Il lato destro del menu pop-up è posto alla posizione X.

Tabella 18.4 *Valori delparametro Flagsper il menupop-up (tipo Behavior).*

Nome costante	Valore	Descrizione
vbPopupMenuLeftButton	O	Una voce del menu pop-up risponde solo utilizzando il pulsante sinistro del mouse. Questo è il valore predefinito.
VbPopupMenuRightButton	2	Una voce del menu pop-up risponde utilizzando sia il pulsante destro sia il sinistro del mouse.

Si può visualizzare solo un menupop-up alla volta. Se volete cambiare il menupop-up visualizzato, per esempio visualizzare i menu di scelta rapida associati ai due pulsanti del mouse, dovete attendere che ilprimo non sia più visualizzato perpoter far comparire il secondo.

Ecco un esempio, dal progetto PopDemo.Vbp, di come sia possibile invocare il genitore del menu pop-up, intitolato *Left Pop-Up* (e chiamto mnuLeft):

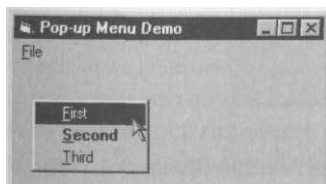
```
PopupMenu mnuLeft,
   vbPopupMenuLeftButton Or vbPopupMenuLeftAlign, _
      , , mnuLeftFirst
```

Questo progetto permette all'utente di decidere quale tra mnuLeft o MnuRight debba essere visualizzato quando viene generato l'evento Click del form .

```
Private Sub mnuContext_Click()
   If mnuContext.Caption = "&Left" Then
      mnuContext.Caption = "&Right"
   Else
      mnuContext.Caption = "&Left"
   End If
End Sub
```

Ed ecco il codice completo per la gestione dell'evento Click del form, il quale visualizza il menu pop-up selezionato dall'utente, come mostrato nella Figura 18.13:

```
Private Sub Form_Click()
   If mnuContext.Caption = "&Left" Then
      PopupMenu mnuLeft, vbPopupMenuLeftButton Or _
         VbPopupMenuLeftAlign,,, mnuLeftFirst
   Else
      PopupMenu mnuRight, vbPopupMenuRightButton Or _
         vbPopupMenuRightAlign,,, mnuRightSecond
   End If
End Sub
```

Figura 18.13
L'input dell'utente determina quale menu pop-up visualizzare.

Gestione dinamica dei menu

Gestione dinamica significa fare in modo che l'aspetto dei menu vari durante resecuzione dell'applicazione (a runtime), generalmente in risposta ad alcune particolari azioni dell'utente. La funzione Window List dei form genitori MDI (vedere la sezione precedente "Come creare applicazioni MDI") è un esempio di gestione dinamica dei menu presente in Visual Basic. Una voce menu viene aggiunta per ogni

form figlio all'interno del progetto MDI non appena viene mostrata. Un esempio di gestione dinamica dei menu che potreste aggiungere a una vostra applicazione potrebbe essere la comparsa di una nuova voce per ogni nuovo documento aperto all'interno della applicazione stessa. Ogni voce di menu ha proprietà Checked Enabled, e Visible che possono essere facilmente modificate in maniera dinamica all'interno del codice.

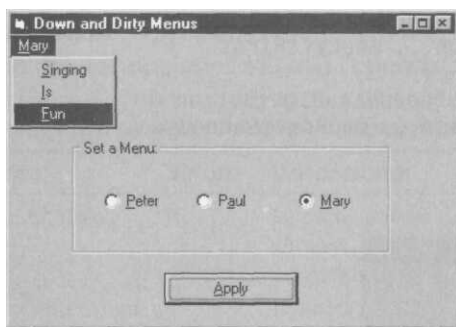
Notate che le voci di menu aventiproprietà Visible posta a False *non sono solo trasformate in invsibili, ma vengono anche automaticamente disabilitate. Questo significa che non dovetepreoccuparvi che l'utentepossa generare eventi che appartengono a voci di menu invisibili.*

Sfruttando le potenzialità native di Visual Basic, opFortunamente estese con l'uso delle API di Windows, potete gestire in maniera molto semplice i menu dinamici.

Visibilità delle voci di menu

Potete controllare la visibilità di tutte le voci di menu e sottomenu sottostanti a un menu di livello superiore con laproprietà Visible *dello stesso menu di livello superiore. Questo significa che si può facilmente rendere alternativamente visibili e invisibili intere strutture di menu infase di esecuzione.*

Come esempio, ho aggiunto una barra di menu al progetto di esempio (salvato come DownDirt.Vbp) con le voci *Peter, Paul* e *Mary.* Ogni menu include anche varie sottovoci. L'idea è che l'utente possa decidere, durante l'esecuzione, quale struttura di menu debba essere visibile (e attiva), come mostrato nella Figura 18.14.

Figura 18.14
Potete usare laproprietà Visible del menu di livello superiore per controllare la visibilità delle strutture dell'interomenu.

Per ottenere questo risultato ho incluso nell'evento Load del form il codice che pone la proprietà Visible di tutti e tre i menu di livello superiore a False:

```
Private Sub Form_Load()
    mnuPeter.Visible = False
    mnuPaul.Visible = False
    mnuMary.Visible = False
End Sub
```

La struttura di menu resa visibile è determinata dalla selezione dell'utente tra una serie di pulsanti di opzione, una volta fatto clic sul pulsante *Apply.*

```
Private Sub cmdApply_Click()
  If optPPM(0) Then          'visualizza il menu di Peter
     mnuPeter.Visible = True
     mnuPaul.Visible = False
     mnuMary.Visible = False
  Elself optPPM(1) Then      'visualizza il menu di Paul
     mnuPeter.Visible = False
     mnuPaul.Visible = True
     mnuMary.Visible = False
  Else    'visualizza     il     menu     di     Mary
     mnuPeter.Visible = False
     mnuPaul.Visible = False
     mnuMary.Visible = True
  End If
End Sub
```

Come eliminare voci menu

// progetto salvato come RemoveM.Vbp mostra come sia possibile rimuovere dinamicamente i menu utilizzando /eAP/GetMenu, GetSubMenu *e* RemoveMenu. *È da notare che sarebbe meglio manipolare le proprietà di visibilità Visual Basic dei menu e delle voci menu anziché usare questa tecnica, la quale, in alcuni casi, può Fortare a errori di esecuzione imprevisti.*

Ecco le dichiarazioni delle API:

```
Private Declare Function GetMenu Lib "user32" _
  (ByVal hwnd As Long) As Long
Private Declare Function RemoveMenu Lib "user32" _
  (ByVal hMenu As Long, ByVal nPosition As Long, _
  ByVal wFlags As Long) As Long
Private Declare Function GetSubMenu Lib "user32" _
  (ByVal hMenu As Long, ByVal nPos As Long) As Long
```

La funzione RemMenu accetta come primo parametro l'handle della finestra contente la barra dei menu, e nei suoi rimanenti argomenti la posizione del menu di livello superiore e del sottomenu che devono essere rimossi. La posizione è specificata partendo da zero, ovvero il menu più a sinistra e più in alto sullo schermo è considerato in posizione O, la sua prima voce menu è in 0,0 e così di seguito. Si usa Get - Menu per riottenere la gestione dell'intero menu collegato alla finestra. GetSubMenu viene invece usata per recuperare l'handle di uno specifico menu di livello superiore. RemoveMenu serve per rimuovere la voce di menu indicata. Il valore restituito dell'API RemoveMenu è assegnato alla funzione RemMenu (si veda il Listato 18.2).

Listato 18.2 *Come rimuovere un menu.*

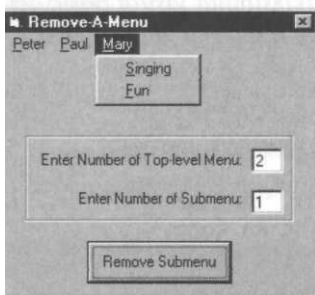
```
Public Function RemMenu(ByVal Handle As Long, _
  ByVal TopPos As Long, ByVal SubPos As Long) As Boolean
  RemMenu = False
  Const MF_BYPOSITION = &H400&
  Dim hMenu As Long, hSubMenu As Long, retval As Boolean
  hMenu = GetMenu(Handle)
  hSubMenu = GetSubMenu(hMenu, TopPos)
```

```
    retval = RemoveMenu(hSubMenu, SubPos, MF_BYPOSITION)
    RemMenu = retval
End Function
```

Ed ecco il codice che chiama la funzione `RemMenu` quando è stato fatto clic sul pulsante *Remove:*

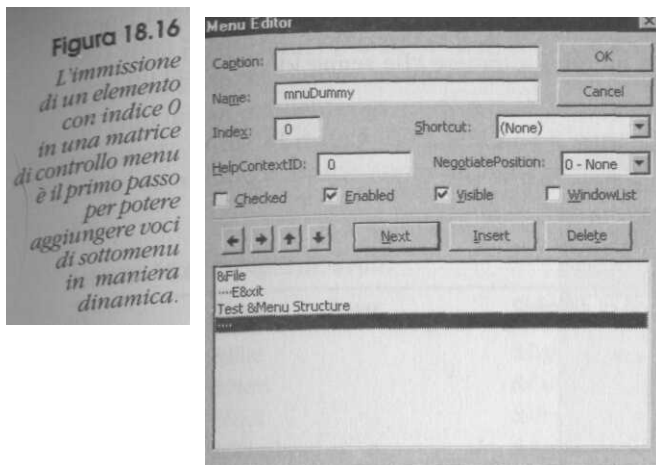
```
Private Sub cmdRemove_Click()
    If Not RemMenu(Me.hwnd, Val(txtTop), Val(txtSub)) Then
        MsgBox "L'eliminazione è fallita!", vbCritical, "Mi dispiace!"
    End If
End Sub
```

Se paragonate il menu mostrato nella Figura 18.14 con quello nella Figura 18.15 noterete che i valori immessi nella Figura 18.15 hanno effettivamente rimosso il menu specificato (il menu di livello superiore *Mary* è in posizione 2; la sua voce menu *Is* è in posizione 1.) Quando lanciate questo programma, ricordatevi solamente che il conteggio dei menu parte da O, e non da 1.

Figura 18.15 *Potete utilizzare l'API RemoveMenu per rimuovere le voci menu.*

Come aggiungere voci di menu

Ecco un segreto tenuto ben custodito: potete aggiungere dinamicamente voci di menu in fase di esecuzione senza uscire da Visual Basic per utilizzare le API. Questa tecnica utilizza l'istruzione Load *per aggiungere nuovi elementi a una matrice dinamica di controllo di voci menu. Il tradizionale utilizzo di* Load *consiste nel caricare un form in memoria senza però mostrarlo (cioè, senza renderlo visibile).*

Per vedere come questofunzioni, diamo un occhiata al sempliceprogetto da me realizzato, presente sul CD-ROM allegato al libro, con il nome Newmenus. Vbp. Ilprimo passo dafare comForta un piccolo trucco con il Menu Editor, come mostrato nella Figura 18.16. File *ed* Exit *sono normali voci del Menu Editor. Il successivo campo del Menu Editor è un menu di livello superiore, intitolato* Test&Menu Structure *e chiamato* mnuTest, *la cuiproprietà* Visible *è stataposta a* False. *Al di sotto di* mnuTest *si trova ilprimo elemento della matrice di controllo. La cosa imFortante da notare e che il valore* caption *di* `mnuDummy` *è stato intenzionalmente omesso.*

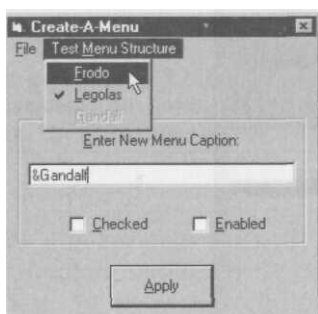
Figura 18.16 *L'immissione di un elemento con indice 0 in una matrice di controllo menu è il primo passo per potere aggiungere voci di sottomenu in maniera dinamica.*

Il codice necessario per aggiungere nuovi sottomenu alla struttura mnuTest è molto semplice, come mostra il Listato 18.3:

Listato 18.3 *Come aggiungere un sottomenu.*

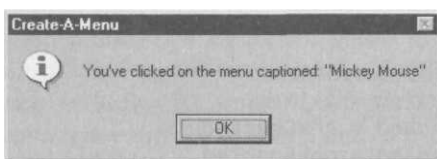
```
Private Sub cmdApply_Click()
   Static ItemCount As Integer
   If txtNew = "" Then
      MsgBox "Non hai inserito una didascalia per il menu!", _
         vbInformation, ProgTitle
      Exit Sub
   Else
      If ItemCount > 0 Then
         Load mnuDummy(ItemCount)
      Else
         mnuTest.Visible = True
      End If
      'Imposta le proprietà del nuovo menu
      mnuDummy(ItemCount).Caption = txtNew
      mnuDummy(ItemCount).Checked = chkChecked
      mnuDummy(ItemCount).Enabled = chkEnabled
      ItemCount = ItemCount + 1
      'Reset txtNew
      txtNew = ""
   End if
End Sub
```

ItemCount tiene traccia del numero di elementi mnuDummy che sono stati aggiunti a mnuTest. Per il primo elemento (elemento 0), non è necessario un caricamento di tipo dinamico poiché il sottomenu esiste già (anche se senza una didascalia). Tuttavia è necessario Forre la visibilità della struttura mnuTest a True la prima volta che viene lanciata la procedura. Una volta che il nuovo elemento è stato aggiunto alla matrice di menu, le sue proprietà (la didascalia e se deve essere marcato e/o attiva-

to) vengono variate in base a ciò che l'utente immette, come mostrato nella Figura 18.17. È da notare che se l'utente immette una "e commerciale" (&) come parte della didascalia per una nuova voce menu, il carattere che segue identificherà un tasto di scelta rapida.

Figura 18.17 *L'istruzione Load è utilizzata per aggiungere dinamicamente voci di menu e questo può comportare anche la generazione dinamica di tasti di scelta rapida.*

È buona norma realizzare un sistema che visualizzi quale voce menu, creata dinamicamente, sia stata attivata. Per farlo (come già spiegato prima in "Matrici di controllo menu" si deve passare il valore Index all'evento Click della matrice di controllo menu. In molti programmi, questo può comFortare l'uso di una istruzione Select Case applicata alla variabile Index; in questo caso, poiché tutto ciò che serve è il valore della stringa didascalia dell'elemento della matrice, si può usare direttamente il valore dell'indice, come mostrato nella Figura 18.18.

```
Private Sub mnuDummy_Click(Index As Integer)
    Const Quote = 34
    MsgBox "You've clicked on the menu captioned: " & _
        Chr(Quote) & mnuDummy(Index).Caption & Chr(Quote), _
        vbInformation, ProgTitle
End Sub
```

Figura 18.18 *Determinazione della voce di menu scelta.*

Per includere i doppi apici all'interno di un letterale stringa, come nella Figura 18.18, bisogna indicarli tramite il relativo codice ASCIl (chr(34)).

Caricamento di stringhe di menu da file esterni

Il Capitolo 16, "Ottimizzazione", mostra come caricare didascalie e icone da un file di risorse esterno. Il testo e le immagini specifiche caricati nell'applicazione di esempio, External.Vbp, dipendono dalla lingua impostata dall'applicazione, decisa da un numero d'offset al momento della compilazione condizionale.

Il caricamento di stringhe di menu esterne funziona nello stesso modo. A meno che non dobbiate caricare dinamicamente voci di menu come descitto nella precedente

sezione dovrete usare il Menu Editor per organizzare una struttura di menu. Dovete assegnare nomi a tutti i menu, ma non è necessario assegnare le didascalie, perché queste saranno comunque sostituite da stringhe esterne.

Ho fatto una copia del progetto External.Vbp, per mostrarvi come funziona il procedimento. Il file di risorsa, Both.Res, che è parte del progetto, è stato ampliato in modo da includere alcune nuove stringhe per i menu, come mostrato nella Tabella 18.5.

Tabella 18.5 *Stringhe di menu.*

ID	Stringa Inglese	Stringa in lingua elfa
105	&File	&Laita
106	&New	&Cuio
107	E&xit	&Pheriain
108	&Window	&Aglar
109	&Help	&Eglerio!

Il codice che imposta il titolo del menu è messo nell'evento Load del form in base a quale costante condizionale è True, come potete vedere nel Listato 18.4.

Listato 18.4 *Caricamento di stringhe di menu dafile esterni.*

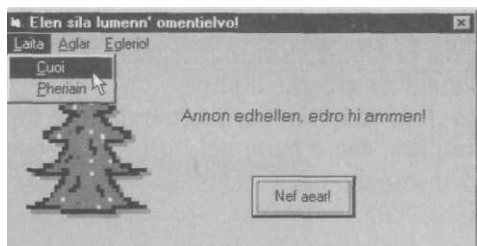
```
Private Sub Form_Load()
   #Const Elvish = True
   #If English Then
      Offset = 100
   #ElseIf Elvish Then
      Offset = 200
   #Else
      MsgBox "Tentativo di caricare una lingua ignota!"
   #End If

   mnuFile.Caption = LoadResString(Offset + 5)
   mnuNew.Caption = LoadResString(Offset + 6)
   mnuExit.Caption = LoadResString(Offset + 7)
   mnuWindow.Caption = LoadResString(Offset + 8)
   mnuHelp.Caption = LoadResString(Offset + 9)
End Sub
```

Il risultato è il menu visibile nella Figura 18.19.
Per Fortare l'applicazione in Unga inglese, sostituite semplicemente

```
#Const English = True
```

al posto di

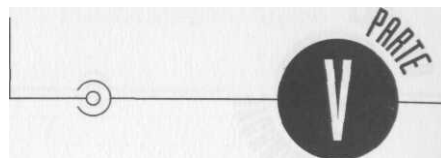
```
#Const Elvish = True
```

Figura 18.19
È facile rendere internazionali i menu caricando stringhe di titoli da file esterni.

La risposta all'attivazione dell'evento menu è gestita nel modo classico, sebbene tutti i letterali stringa debbano essere caricati esternamente utilizzando gli offset (scostamenti). Per esempio:

```
Private Sub mnuHelp_Click()
   MsgBox LoadResString(Offset + 4), _
      vbInformation,
      LoadResString(Offset)
End Sub
```

Riepilogo

È facile creare una struttura per la gestione delle applicazioni MDI, se si seguono le tecniche spiegate in questo capitolo.

- Avete imparato come tenere traccia dei form figli MDI.
- Avete imparato come mostrare form secondari MDI da un form modale.
- Ho spiegato come utilizzare gruppi di form per minimizzare o chiudere tutti i form figli.
- Ho mostrato come aggiungere sfondi a un'applicazione MDI.
- Avete scoperto come creare sfondi ripetendo più volte la stessa immagine.
- Avete imparato come dimensionare i form figli nell'evento Load del form.
- Avete imparato come impostare un cursore personalizzato.
- Avete imparato a utilizzare il Menu Editor.
- Avete imparato come impostare i nomi di menu.
- Ho spiegato come utilizzare la proprietà NegotiatePosition .
- Avete scoperto le matrici di controllo menu.
- Ho mostrato come creare menu pop-up.
- Avete imparato come controllare la visibilità dei menu durante l'esecuzione dell'applicazione.
- Avete imparato come rimuovere voci di menu in fase di esecuzione.
- Ho spiegato come aggiungere voci di menu in fase di esecuzione.
- Ho discusso il caricamento di stringhe di menu da un file esterno.

USO DI ACTIVEX

VISUALIZZAZIONE DURANTE L'ESECUZIONE

- Tecniche per personalizzare l'aspetto di applicazioni Visual Basic
- Come creare "uova di Pasqua"
- La "vita segreta" dei Form Visual Basic

L'aspetto delle applicazioni influenza molto il modo in cui l'utente le percepisce. Questo capitolo tratta varie tecniche per aggiungere abbellimenti e perfezionamenti visivi. Vi ricordo nuovamente che meno è meglio: non esagerate con questi effetti. (Per maggiori informazioni in proposito, si veda il Capitolo 17.)
Le "uova di Pasqua" *(Easter eggs)* sono "sorprese" nascoste all'interno di qualche elemento di un'applicazione. Di solito mostrano informazioni circa i creatori del programma. Un esempio è l'uovo di Pasqua di Windows 95. In questo capitolo, vi mostrerò come creare delle sorprese di questo genere. Il capitolo si conclude con una carrellata sulla vita "segreta" dei form VB: dimostrerò come manipolare form VB e file di progetto come testo in formato ASCIl!

Effetti speciali

Dal punto di vista dell'utente, *Visual* Basic ha a che fare con gli effetti *visivi* di un programma. A patto di non eccedere, niente può dare maggior successo alle vostre applicazioni di qualche effetto speciale interessante. Come vi mostrerò, è facile creare visualizzazioni di grande qualità senza usare strumenti di terze parti. (Potete utilizzare strumenti esterni invece per aggiungere facilmente un aspetto caratteristico alle applicazioni).

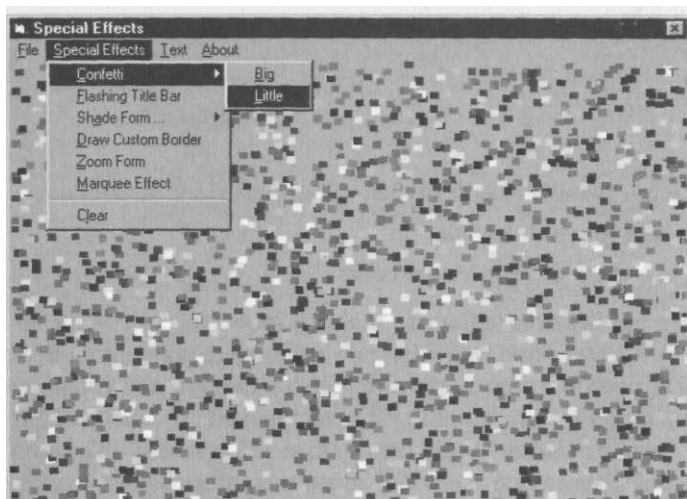
Gli effetti grafici spettacolari, che imparerete a creare in questo capitolo, sono tutti contenuti in una sola applicazione, presente sul CD-ROM allegato al libro, con il nome di Effects.Vbp. (Potrete accedere agli effetti forniti dall'applicazione con il menu Special Effects.)

Tutte le procedure utilizzate sono contenute in un unico modulo di codice, FxCode.Bas. Potete semplicemente aggiungere questo modulo al vostro progetto e richiamare le routine contenute passando i form del progetto come argomenti. Il codice di molti effetti speciali è troppo lungo per essere riFortato interamente nel

testo, ma è tutto riportato nel CD-ROM allegato al libro. Ovviamente spiegherò come funzionano le procedure fondamentali.

Coriandoli

L'effetto "coriandoli" consiste in quadrati policromi che lampeggiano in maniera casuale sullo schermo, come mostrato nella Figura 19.1.

Figura 19.1 *Potete facilmente aggiungere alla vostra applicazione un efetto confetti con colori casuali.*

La routine Confetti utilizza la funzione QBColor per ottenere valori di colore RGB casuali. I valori di colore RGB (come vengono forniti dalla funzione RGB usata dal Visual Basic) sono un unico numero (di tipo long integer) che rappresenta le componenti di rosso,verde e blue dei colori specifici da visualizzare. I valori di intensità delle singole componenti vanno da O a 255. I valori RGB restituiti dalla funzione QBColor rappresentano solo un sottoinsieme delle combinazioni poiché sono possibili solo 15 valori di ritorno da QBColor. Il metodo Line del form è utilizzato per disegnare, in questo caso, i quadratini dei coriandoli in posizioni casuali sul form. (La direttiva facoltativa B alla fine della chimata del metodo Line disegna un quadratino; la F specifica che deve essere riempito.)
La dimensione dei quadratini dipende dall'argomento ScaleMode passato alla procedura. La Figura 19.1 mostra piccoli coriandoli con il parametro Size impostato a vbTwips. Potete creare coriandoli più grandi dando al parametro Size il valore vbPixels. La proprietà ScaleMode del form, come noterete, è ripristinata al valore originale alla fine della procedura, come si può vedere nel Listato 19.1.

Listato 19.1 *Comegenerare "coriandoli".*

```
Public Sub Confetti(Frm As Form, Size As Integer)
   Dim I As Integer, X1 As Integer, Y1 As Integer, _
      Color As Long, OldScaleMode As Integer
   OldScaleMode = Frm.ScaleMode
   Frm.ScaleMode = Size
   Randomize
   ForI=1To100
      X1 = Rnd * Frm.ScaleWidth
      Y1 = Rnd * Frm.ScaleHeight
      Color = QBColor(Rnd * 15)
      Frm.Line (X1, Y1)-(X1 + 85, Y1 + 65), Color, BF
   Next  I
Frm.ScaleMode = OldScaleMode
End Sub
```

La procedura Confetti può essere chiamata da un Timer di controllo. (Nel programma dimostrativo, la proprietà Interval del Timer è posta a uno.) Abilitate il Timer quando volete che i coriandoli cadano; disabilitatelo quando volete che si fermino.

Come far lampeggiare la barra del titolo

È semplice far lampeggiare la barra del titolo di una finestra mediante la funzione API FlashWindow. Questo effetto è accattivante per l'utente e certamente attirerà la Sua attenzione.

Come per tutti gli effetti speciali, è buona norma non abusarne. Un 'applicazione con troppe barre lampeggiantifarebbe impazzire l'utente.

Ecco la dichiarazione dell'API:

```
'Dichiarazione API per FlashWindow
Declare Function FlashWindow Lib "user32" (ByVal hwnd As Long, _
   ByVal bInvert As Long) As Long
```

Per far lampeggiare la barra del titolo impostate il secondo parametro a True.

// valore di ritorno della funzione FlashWindow *non indica, come ci si potrebbe aspettare, il successo o ilfallimento dell'effetto. Il valore indica invece se la finestra passata era attiva prima della chiamata allafunzione.*

Come potrete vedere eseguendo il programma di esempio, il titolo continuerà a lampeggiare anche nella barra delle applicazioni di Windows 95, se la finestra viene ridotta a icona. Potreste aggiungere una barra del titolo lampeggiante a un'applicazione del "vassoio", rappresentata da un'icona (nel Capitolo 11 troverete informazioni su come creare applicazioni per il "vassoio") per attirare l'attenzione, per esempio quando si verifica una condizione di emergenza. Per far continuare il lampeggiamento, dovrete richiamare la API da un controllo Timer abilitato, come indicato nel Listato 19-2.

Listato 19.2 *Come far lampeggaire la barra del titolo.*

```
Private Sub tmrFlash_Timer()
   Call FXCode.FlashTitle(Me.hwnd, True)
End Sub

'Chiamata API
Public Sub FlashTitle(Handle As Long, ReturnOrig As Boolean)
   Call FlashWindow(Handle, ReturnOrig)
End Sub
```

Come sfumare un form

Far sfumare un form, facendolo passare da chiaro a scuro, è un effetto comune ma sempre notevole. Di solito, ciò vuoi dire che lo sfondo è più chiaro in cima e più scuro sul fondo, ma potete ovviamente cambiare il codice d'esempio per adattarlo alle vostre esigenze.
Un'applicazione, che avrete sicuramente visto, che utilizza form sfumati è il programma di setup di Windows, che parte da un blu chiaro in alto per terminare nel blu scuro del fondo. Il Listato 19.3 contiene, nella procedura ShadeForm, il codice che disegna uno sfondo sfumato rosso, verde o blu sul form passato come argomento.

Listato 19.3 *Come sfumare un form.*

```
'Salva le impostazioni correnti di form/modalità
DS = frm.DrawStyle
DW = frm.DrawWidth
SM = frm.ScaleMode
SH = frm.ScaleHeight
'settings for shading
frm.DrawStyle = vbInsideSolid
frm.DrawWidth = 2
frm.ScaleMode = vbPixels
frm.ScaleHeight = 256
For I = 0 To 255
   Select Case TheColor 'Sfuma il form in base al colore passato
      Case ()
         frm.Line (0, I)-(frm.Width, I + 1), _
            RGB(255 - I, 0, 0), B ' red
      Case 1
         frm.Line (0, I)-(frm.Width, I + 1), _
            RGB(0, 255 - I, 0), B 'green
      Case 2
         frm.Line (0, I)-(frm.Width, I + 1), _
            RGB(0, 0, 255 - I), B 'blue
      Case Else
         MsgBox "Errore interno nella selezione del colore!"
   End Select
```

```
Next I
  'Ripristina le impostazioni originali di form/modalità
frm.DrawStyle = DS
frm.DrawWidth = DW
frm.ScaleHeight = SH       'deve essere ripristinata prima di ScaleMode
frm.ScaleMode = SM
```

La proprietà DrawStyle del form controlla lo stile delle linee prodotte dai metodi grafici (come il metodo Line chiamato in seguito nella procedura). DrawWidth imposta la *larghezza,* dell'output del metodo Line a 2 pixel. ScaleMode imposta l'unità di misura del sistema di coordinate della form, come indicato nella Tabella 19.1.

Tabella 19.1 *Valori della proprietà ScaleMode.*

Inpostazione	Costante	Commento
0	vbUser	Questa è l'impostazione personalizzata, nel senso che una o più delle proprietà ScaleHeight, ScaleLeft, ScaleTop, o ScaleWidth è stata impostata a un valore non di default.
1	vbTwips	Valore di default (un pollice = 1440 twip).
2	vbPoints	I punti sono una unità di misura tipografica (un pollice verticale = 72 punti).
3	vbPixels	Un pixel rappresenta l'unità minima di risoluzione del monitor o della stampante.
4	vbCharacter	Orizzontale = 120 twip, e Verticale= 240 twip per "carattere"
4	vbInches	
	vbMillimeters	
6	vbCentimeters	

Probabilmente sarete abituati a leggere le proprietà ScaleHeight e ScaleWidth del form per avere l'unità di misura e per trovare l'area client di un form. Potete anche utilizzare ScaleHeight e ScaleWidth per impostare un sistema di coordinate interno al form; qui ScaleHeight è utilizzata per impostare a 256 il numero di linee verticali sullo sfondo della form. Un ciclo For. . .Next richiama il metodo Line tante volte quante sono le righe.
Il metodo Line disegna un riquadro alto ScaleHeight (è specificata la direttiva B) su ogni "linea" del form.La *larghezza* del riquadro è la *larghezza* client del form. Il gradiente di colore è ottenuto dall'aumento in maniera incrementale del valore primario della funzione RGB, mediante l'indice del ciclo. Per esempio, blu è il terzo argomento di RGB, e la varibile I varia da O fino a 255 con passo 1 (ogni volta disegna 1/256 dello sfondo del form):

```
RGB(0,0, 255 - I), B 'blue
```

La Tabella 19.2 mostra gli equivalenti della funzione RGB per i colori più comuni.

Tabella 19.2 *7 colori equivalenti dettafunzione RGB.*

Colore	Rosso (R)	Verde (G)	Blu (B)
Nero	0	0	0
Blue	0	0	255
Verde	0	255	0
Cyan	0	255	255
Rosso	255	0	0
Magenta	255	0	255
Giallo	255	255	0
Bianco	255	255	255

Come disegnare i contorni di un form

Potete utilizzare il metodo Line per disegnare tante linee quante ne volete intorno ai contorni di un form. L'effetto, mostrato nella Figura 19.2, sembra una cornice "a gradini" per un quadro. Il Listato 19.4 riporta il codice necessario.

Figura 19.2 *Potete utilizzare il metodo Line per adattare l'aspetto del vostro form.*

Listato 19.4 *Per disegnare i contorni di unform.*

```
Public Sub DrawBorder(frm As Form, NumLines)
   Dim I As Integer, Restore As Boolean, OldScaleMode As Integer
   Const DrawWidth = 2
   OldScaleMode = frm.ScaleMode
   frm.ScaleMode  =  vbPixels
   Restore = frm.AutoRedraw
   frm.AutoRedraw = True
   For I = DrawWidth - 1 To NumLines * 3 Step 3
      frm.Line (I, I)-(frm.ScaleWidth - 1 - I, _
         frm.ScaleHeight - 1 - I), , B
   Next I
```

```
   frm.ScaleMode = OldScaleMode
   frm.AutoRedraw = Restore
End Sub
```

Impostate il metodo AutoRedraw *del form a* True *(di default è False) prima di richiamare il metodo* Line. *Ciò significa che ilform è ridisegnato da una copia in memoria anziché essere creato dall'evento* Paint *del form.*

Se volete disegnare i contorni in un colore diverso dal nero, potete semplicemente impostare laproprietà ForeColor *prima di applicare il metodo* Line. *Per esempio:*

```
frm.ForeColor = vbYellow
```

Come far esplodere un form

Far *esplodere* un form (qualche volta si parla anche di *zoom)* vuoi dire farlo comparire rapidamente, come in un'esplosione, dal suo punto centrale. Potete creare l'effetto nascondendo, nel codice, il form e disegnando in successione rettangoli, del colore del form, sempre più grandi. Quando i rettangoli sono della stessa dimensione del form, questo viene visualizzato e lo zoom si ferma.
La velocità apparente dell'esplosione è controllata dal numero di rettangoli disegnati. (Dice il saggio: Far esplodere un form è un bel modo per attirare l'attenzione e agli utenti la cosa iace molto, ma non tirate per le lunghe anche il processo.) Se volete potete fare lo zoom da un punto diverso da quello centrale, ma dovrete modificare il codice fornito. Disegnare direttamente sullo schermo (anziché sul form) richiede l'uso di alcune funzioni API. Il Listato 19.5 contiene le dichiarazioni.

Listato 19.5 *Dichiarazioni API di form che esplode.*

```
Type RECT
       Left As Long
       Top As Long
       Right As Long
       Bottom As Long
End Type
Declare Function CreateSolidBrush Lib "gdi32" _
   (ByVal crColor As Long) As Long
Declare Function DeleteObject Lib "gdi32" _
   (ByVal hObject As Long) As Long
Declare Function GetDC Lib "user32" (ByVal hwnd As Long) As Long
Declare Function GetWindowRect Lib "user32" _
   (ByVal hwnd As Long, _
   lpRect As RECT) As Long
Declare Function ReleaseDC Lib "user32" (ByVal hwnd As Long, _
   ByVal hdc As Long) As Long
Declare Function SelectObject Lib "gdi32" (ByVal hdc As Long, _
   ByVal hObject As Long) As Long
Declare Function Rectangle Lib "gdi32" (ByVal hdc As Long, _
   ByVal X1 As Long, ByVal Y1 As Long, _
   ByVal X2 As Long, ByVal Y2 As Long) As Long
```

La procedura ExplodeForm *funziona ricavando le dimensioni e la posizione del form corrente dalla chiamata* API GetWindowRect *con il forra trattato come argomento:*

```
GetWindowRect frm.hwnd, ThisRect
RectWidth = (ThisRect.Right - ThisRect.Left)
RectHeight = ThisRect.Bottom - ThisRect.Top
```

Poi si ricava un handle di dispositivo, vale a dire un riferimento, per lo schermo, si crea un pennello Windows basato sul colore del form e si salva una copia del vecchio pennello:

```
ScreenDevice = GetDC(0)
NewBrush = CreateSolidBrush(Color)
OldBrush = SelectObject(ScreenDevice, NewBrush)
```

Infine, vengono disegnati i rettangoli sullo schermo mediante Phandle del contesto di dispositivo schermo e l'API Rectangle:

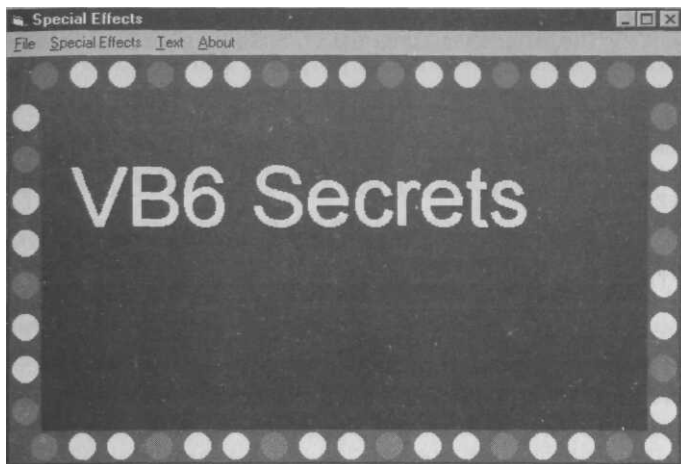
```
For I = 1 To Steps
   XRect = RectWidth * (I / Steps)
   YRect = RectHeight * (I / Steps)
   X = ThisRect.Left + (RectWidth - XRect) / 2
   Y = ThisRect.Top + (RectHeight - YRect) / 2
   'Disegna incrementalmente il rettangolo
   Rectangle ScreenDevice, X, Y, X + XRect, Y + YRect
Next I
```

È importante ripristinare il vecchio pennello, cancellare il gestore screen device, ed eliminare il pennello che era stato usato per disegnare i rettangoli:

```
Call SelectObject(ScreenDevice, OldBrush)
Call ReleaseDC(0, ScreenDevice)
DeleteObject (NewBrush)
```

Come creare un effetto Marquee

Un effetto *marquee,* una forma di evidenziazione costituita da figure in movimento che sembrano accese al bordo di un form, attira abbastanza l'attenzione (come potete vedere nella Figura 19.3).
La procedura Marquee è richiamata da un controllo Timer (per ottenere i migliori risultati, impostate la proprietà Interval tra 50 e 250). In questo modo verrà richiamata più volte la procedura dando un apparenza di movimento. Il nocciolo dell'effetto sta nel disegnare le figure nelle posizioni appropriate sul form. Questo è realizzato nella procedura DrawShape, che usa i metodi Circle e Line per creare un totale di otto differenti tipi di figure sul form. Due di queste figure sono state realizzate con il codice riportato nella pagina seguente.

Figura 19.3 *Potete utilizzare il codice del modulo FXCode.Bas per aggiungere effetti marquee in movimento alle applicazioni.*

```
'Cerchio
frm.Circle (HorizontalSpread, VerticalSpread), Size, FillColor

'Quadrato
frm.Line (HorizontalSpread - (Size \ 2), _
   VerticalSpread - (Size \ 2))-(HorizontalSpread + (Size \ 2), _
      VerticalSpread + (Size \ 2)), FillColor, BF
```

DrawShape è richiamata ripetutamente dalla procedura Marquee per disegnare la figura voluta in alto, in basso, a destra e a sinistra del forni.
Per esempio, ecco il codice per farlo in alto e a destra del forni:

```
' La parte superiore
For I = 1 To Across

   DrawShape frm, WhichShape, FillColor, BackColor,_
      HorizontalSpread,VerticalSpread,FormScaleHeight,_
      FormScaleWidth

Next I

   ' Sul lato destro
For I = 1 To Down - 1
   DrawShape frm, WhichShape, FillColor, BackColor,
      HorizontalSpread, VerticalSpread, FormScaleHeight, _
      FormScaleWidth
```

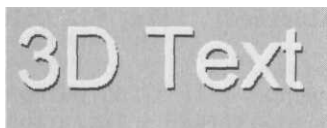
Per vedere il codice completo fate riferimento al modulo FXCode.Bas sul CD-ROM allegato al libro.

Stampa di testo tridimensionale sul form

È facile utilizzare il metodo Print e le proprietà dell'oggetto Font del form, per visualizzare direttamente testo sul form. Per esempio:

```
frm.Font.Name = "Arial"    ' seleziona un tipo di carattere
frm.Font.Italie = True     ' imposta le proprietà del tipo di carattere
frm.Font.Size = 48
frm.CurrentX = 200         '  posiziona il cursore
frm.CurrentY = 100
frm.Print "from IDG Books"  ' visualizza il testo
```

Il colore del testo dipende dall'impostazione corrente della proprietà ForeColor del form, che potete impostare nel codice all'esecuzione.
Per vivacizzare un po' la visualizzazione del testo potete farlo sembrare tridimensionale aggiungendo successive chiamate al metodo Print, modificando leggermente la posizione del testo. Per esempio, la procedura PrintName visualizza testo in giallo con il fianco sinistro chiaro e quello destro scuro (Figura 19.4).

Figura 19.4 *Potete far sembrare il testo tridimensionale.*

Come potete osservare nel Listato 19.6, PrintName pone il testo indicato sul form e nella posizione iniziale passata come argomento.

Listato 19.6 *Visualizzazione di testo tridimensionale su unform.*

```
Sub PrintName(frm As Form, TheName As String, _
   X As Integer, Y As Integer)
   'X,Y sono le coordinate del punto iniziale di stampa
   Dim OldScaleMode As Integer, OldForeColor As Long
   OldScaleMode = frm.ScaleMode
   OldForeColor = frm.ForeColor
   If frm.ScaleMode <> vbTwips Then frm.ScaleMode = vbTwips
   frm.ZOrder 0   'necessario   per stampare su colori disegnati
   frm.ForeColor = RGB(32, 32, 32)'grigio molto scuro per l'ombra
   frm.Font.Name = "Arial"
   frm.Font.Size = 48
   frm.CurrentX = X              'posiziona il cursore
   frm.CurrentY = Y
   frm.Print TheName              'visualizza il testo
   frm.ForeColor = RGB(255, 255, 255)  'bianco per evidenziare
   frm.CurrentX = X - 35 'posiziona l'evidenziazione (sinistra in alto)
   frm.CurrentY = Y - 45
   frm.Print TheName          'stampa l'evidenziazione in bianco
   frm.ForeColor = vbYellow   'imposta a giallo standard
   frm.CurrentX = X - 25      'visualizza fra evidenziazione e ombra
   frm.CurrentY = Y - 35
   frm.Print TheName          'stampa giallo
```

```
   frm.ScaleMode = OldScaleMode 'ripristina ScaleMode!
   frm.ForeColor = OldForeColor  'ripristina ForeColor
End Sub
```

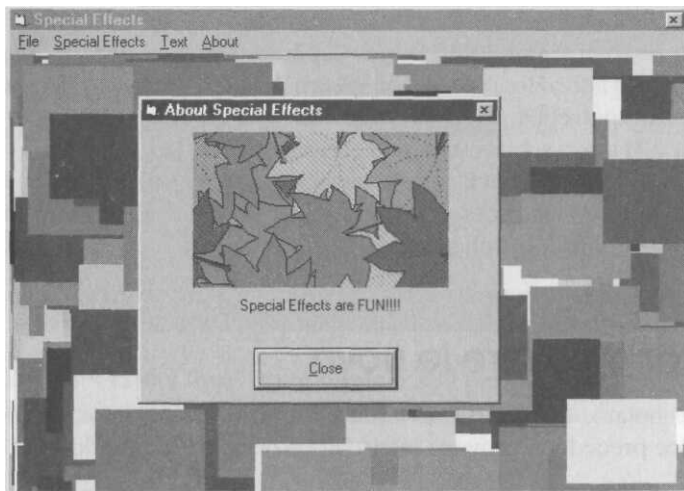
Come mettere "uovo di Pasqua" nel vostro programma

Un uovo di Pasqua, o *Easter egg,* è una visualizzazione "a sorpresa" incorporata in un'applicazione, attivata da qualche azione dell'utente segreta, o non particolarmente segreta. Spesso, la sorpresa è costituita dal nome del creatore del programma stesso. Le uova di Pasqua sono l'equivalente software delle targhe personalizzate per le automobili. Vale a dire: non hanno nessuna utilità se non quella di solleticare la vanità del proprietario. (Dobbiamo confessare che le nostre auto hanno targhe personalizzate!)

Le uova di Pasqua possono essere affermazioni di orgoglio dell'autore del software. Di per sé possono servire come un incentivo morale per gruppi, complessi e movimentati, di sviluppatori. Benché non siano di rilievo pratico, sono divertenti da pensare e aiutano a capire alcune utili tecniche di programmazione.

Come "deporre" un uovo

In genere, le uova sono deposte (vale a dire attivate) da alcune sequenze di movimenti del mouse o da input da tastiera. Il progetto, che mostra come aggiungere effetti speciali ai form (Effects.Vbp), ha una finestra *About Sepecial Effects* (mostrata nella Figura 19.5) che contiene una sorpresa.

Figura 19.5 *La finestra di dialogo About Special Effects contiene una sorpresa.*

Per attivare l'uovo, fate clic sulla bitmap raffigurante delle foglie autunnali nella finestra *About Special Effects.* Poi, premete contemporaneamente Ctrl, Maiusc e F10.

L'uovo comincerà ad schiudersi, vale a dire che un controllo invisibile Picture contenente una bitmap di nuvole, verrà ridimensionato e visualizzato in maniera tale da occupare l'intera finestra *About* del form. Poi cominciano a scorrere lungo il controllo Picture dei banner (in effetti sono etichette).
Il codice che attiva questo uovo è posto nell'evento KeyDown del controllo Picture che contiene la bitmap delle foglie autunnali. (Affinchè il control Picture possa ricevere input da tastiera per l'elaborazione, deve avere il focus, ed è per questo che bisogna farci sopra un clic.)

Listato 19.7 *Come attivare un uovo con un input da tastiera.*

```
PrivateSubPicture1_KeyDown(KeyCodeAsInteger,_
   Shift As Integer)
   Dim ShiftDown As Boolean, CtrlDown As Boolean
   ShiftDown = (Shift And vbShiftMask) > 0
   CtrlDown = (Shift And vbCtrlMask) > 0
   If KeyCode = vbKeyF10 Then
      If ShiftDown And CtrlDown Then     ' Apri quell'uovo!
         pctEgg.Height = Me.ScaleHeight
         pctEgg.Width = Me.ScaleWidth
         pctEgg.Visible = True
         Timer1 .Enabled = True
      End If
   End If
End Sub
```

Chiaramente potete aggiungere molti fattori di complessità diversi all'attivazione delle uova. È possibile monitorare l'evento KeyDown di molti controlli diversi e richiedere che il focus sia passato per quei controlli in una data sequenza, prima che venga registrato l'input da tastiera per quel controllo. Un'altra possibilità è quella di utilizzare una matrice dinamica, una stringa, o una struttura di stack (come quella sviluppata nel Capitolo 13 e nel Capitolo 14) per "registrare" gli input da tastiera; l'evento d'attivazione potrebbe essere realizzato da più input separati. (Per esempio, l'utente potrebbe dover premere i tasti Ctrl e Maiusc, scrivere CAPTAIN e rilasciare Ctrl e Maiusc, schiacciare Alt e scrivere PICARD.)
Se volete essere propri sadici, potete richiedere che qualsiasi azione dell'utente venga effettuata in un lasso di tempo specifico (pena il dover ricominciare tutto da capo). Per questo, usate la funzione API GetTickCount, della quale abbiamo parlato nel Capitolo 16 .

Come far muovere le uova

Come avrete notato, un'attivazione andata a buon fine di un uovo descritta nelle linee di codice precedenti provoca la visualizzazione del controllo Picture e abilita un timer:

```
' Sono stato attivato!
pctEgg.Height = Me.ScaleHeight
```

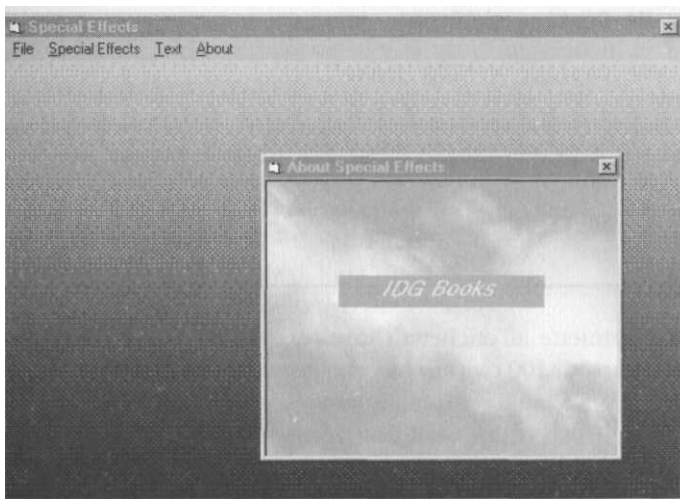
```
pctEgg.width=Me.ScaleWidth
    pctEgg.Visible =True
Timer1.Enabled = True
```

Notate che il codice per ridimensionare pctEgg all'area client del form potrebbe ssere posto altrove (per esempio, quando si carica il form), perché pctEgg rimane comunque invisibile. Le righe importanti del codice sono quelle che abilitano il timer e rendono visibile pctEgg.

per impostare ciò, dovrete aggiungere un controllo Picture (come pctEgg) al form nella fase di progettazione con la sua proprietà Visible impostata a False. La visibilità di tuti i controlli, situati al di sopra di pctEgg, è gestita dalla proprietà Visible di pctEgg.

Quando collocate i controlli su unform, può essere difficoltoso accedere a quelli sottostanti in fase di progettazione. Per poterlo fare, selezionate il controllo (utilizzando, se necessario la finestra Properties) *e scegliete* Bring to Front *(Ctrl+J) dall'opzione* Order *nel menu VB* Format *(o dal menu di scelta rapida).*

Nel programma dimostrativo, su pctEgg viene posta un'etichetta, chiamata MoveMe. Il codice nel controllo timer sposta l'etichetta verso il basso e cambia il contenuto della didascalia. L'effetto risultante è un testo sul video che si muove verso il basso della finestra. La Figura 19.6 mostra una di queste etichette derivanti da questa sequenza animata. (Ho aggiunto una bitmap di nuvole alla proprietà Picture di pctEgg per dare una sensazione eterea.)

Figura 19.6 *Questo uovo di Pasqua visualizza del testo in movimento nel cielo.*

La proprietà Interval del timer è impostata a 300 per permettere all'utente di leggere la didascalia durante il movimento. Il Listato 19.8 mostra il codice dell'evento timer.

Listato 19.8 *Come visualizzare un uovo in movimento.*

```
Private Sub Timer1_Timer()
   Static Flag As Integer
   Dim MoveBy As Integer
   'scalemode è in twip!
   If MoveMe.Top > Me.ScaleHeight Then
      MoveMe.Top = 0
   Else
      MoveBy = -100
   End If
   MoveMe.Move MoveMe.Left, MoveMe.Top - MoveBy
   'Debug.Print MoveMe.Top
   Select Case Flag
      Case 0
         MoveMe.Caption = "Thank you"
      Case 1
         MoveMe.Caption = "for"
      Case 2
         MoveMe.Caption = "reading"
      Case 3
         MoveMe.Caption = "Visual Basic 6"
      Case 4
         MoveMe.Caption = "Secrets"
      Case 5
         MoveMe.Caption = "by Harold Davis"
      Case 6
         MoveMe.Caption = "from"
      Case 7
         MoveMe.Caption = "IDG Books"
   End Select
   Flag = Flag + 1
   'reimposta a 0 la variabile flag all'ottavo ciclo
   If Flag = 8 Then
      Flag = 0
   End If
End Sub
```

Questo codice permette all'etichetta (MoveMe), che parte dall'alto verso il basso di pctEgg , di spostarsi di 100 twip in basso ogni volta che l'evento è attivato.

```
MoveBy = -100
```

```
MoveMe.Move MoveMe.Left, MoveMe.Top - MoveBy
```

Ricordatevi che quando MoveMe si muove verso il basso, il numero rappresentato da MoveMe.Top aumenta. Quando MoveMe raggiunge la fine del form viene riportata in cima:

```
If MoveMe.Top > Me.ScaleHeight Then
      MoveMe.Top = 0
```

potrebbe essere difficile determinare quali cambiamenti apportare al codice che muove i controlli. L'uso di Debug. Print *vi può aiutare a tener traccia delle coordinare del controllo guardando il contenuto della finestra* Debug *Per esempio,* Debug. Print MoveMe.*Top fornisce costantemente il valore di* MoveMe. Top *quando le uova sono attive.*

La variabile statica Flag è utilizzata per variare continuamente tra otto possibili valori della didascalia di MoveMe. Ogni volta che si attiva il codice del timer, Flag è incrementata di 1; quando raggiunge 7 ritorna a 0.
Questo uovo pasquale di dimostrazione, semplice com'è, funziona abbastanza bene ed è piuttosto elegante. Ovviamente, la bellezza e la complessità delle vostre uova pasquali dipendono solo dal tempo che avrete a disposizione, dalla vostra ingegnosità e immaginazione. Molte tecniche descritte precedentemente in questo capitolo, per esempio l'effetto coriandoli, sono perfette per le uova.

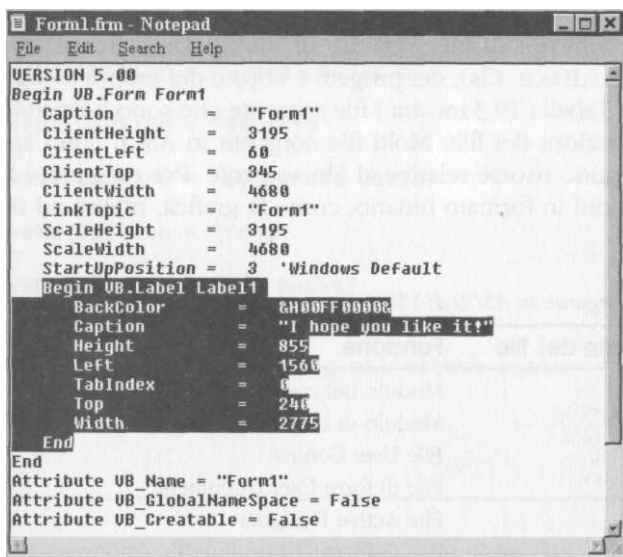
La vita segreta dei form

Fa comodo che molti tipi di file sorgente di Visual Basic, fra cui i file dei form (.Frm), dei moduli (,Bas e .Cls), dei progetti (.Vbp) e dei gruppi (.Vbg), siano semplici file ASCII. La Tabella 19.3 mostra i file sorgente che sono formattati come semplice testo e le funzioni dei file. Molti file sorgente in ASCII sono associati a file binari che contengono risorse relative al file sorgente. Per esempio, i file .Frx contengono informazioni in formato binario, come la grafica, riferite ad un modulo di un form (.Frm).

Tabella 19.3 *File sorgente in ASCII di VB6.*

Estensione del nome del file	Funzione
.Bas	Modulo del codice
.Cls	Modulo di classe
.Ctl	File User Control
.Dob	File di form User Document
.Dsr	File Active Designer
.Frm	File di form
.Pag	File Property Page
.Vbg	Progetto di gruppo Visual Basic
.Vbp	Progetto Visual Basic

II fatto che il codice sorgente di moduli e progetti Visual Basic sia in formato ASCII consente di automatizzare facilmente l'analisi del progetto e del form: un programma, come un add-in, può aprire un form, analizzare e modificare il suo testo ASCII che rappresenta il form da sistemare, e salvare la versione modificata. Questo significa che potete modificare manualmente file di form e progetti utilizzando un qualsiasi editor ASCII (per esempio Blocco note). Allo stesso tempo, sempre come esempio, Potete aggiungere più facilmente le stesse strutture di menu a più form.

All'interno dei form

Le informazioni all'interno dei form sono *organizzate* gerarchicamente. Potete indicare che un oggetto sia il figlio di un altro poiché il sotto-oggetto è rientrato a destra (il testo selezionato nella Figura 19.7 rappresenta una etichetta su un form). Gruppi logici sono racchiusi da istruzioni Begin ed End. Sia VB5 sia VB6 fanno iniziare il testo del file di un form con la stessa indicazione di versione di Visual Basic ("VERSION 5.00") come mostrato nella Figura 19.7.
Gli oggetti e il codice sono indicati nel seguente ordine:

- Proprietà Form
- Proprietà Control
- Struttura menu
- Dichiarazioni generali
- Procedure

```
Form1.frm - Notepad
File  Edit  Search  Help
VERSION 5.00
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   3195
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   4680
   LinkTopic       =   "Form1"
   ScaleHeight     =   3195
   ScaleWidth      =   4680
   StartUpPosition =   3  'Windows Default
   Begin VB.Label Label1
      BackColor       =   &H00FF0000&
      Caption         =   "I hope you like it!"
      Height          =   855
      Left            =   1560
      TabIndex        =   0
      Top             =   240
      Width           =   2775
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
```

Figura 19.7
Il segreto dei form Visual Basic è che sono realmente file in formato ASCII; il testo mostrato include un controllo etichetta on il titolo "I hope you like it!"

Le proprietà dei controlli e le strutture di menu possono contenere blocchi Begin . . . End annidati. Per esempio, un sottomenu è inglobato all'interno di un blocco menu; una etichetta posta su un controllo Picture sarà inserirla all'inteno del controllo Picture. Il testo mostrato nella Figura 19-7 è una semplice descrizione ASCII di un form con un unico controllo etichetta intitolato "I hope you like it!". La proprietà BackColor dell'etichetta è posta a Blue (so che &H00FF0000& corrisponde a blu, perché ho impostato l'etichetta in fase di progettazione).
Potete facilmente modificare le proprietà dell'etichetta cambiando BackColor a 65535 (giallo), il titolo a "Yes!" e l'altezza a 695. La prossma volta che aprirete il progetto contenente il form, verranno riportate le modifiche da voi apportate.

Non capita di rado che si voglia copiare la struttura di un menu da un forni all'altro. Ciò è facilmente realizzabile utilizzando *Copia* e *Incolla* sul form in formato ASCII, ma ci vuole molto tempo se lo si fa con il Menu Editor di Visual Basic. Per esempio, supponete di volere copiare il menu *Special Effects* da Effects.Frm(frmFX) a un altro forni. L'inizio della struttura del menu è mostrato nella Figura 19.8.
Se copio questa struttura negli Appunti (facendo attenzione a inglobare tutto quello che compare tra Begin ed End corrispondenti) e la incollo in un nuovo form, essa mostrerà la struttura di menu copiata nella sua interezza quando verrà caricata, come mostrato nella Figura 19.9.

Figura 19.8
I menu sono organizzati gerarchicamente quando guardate il form informato ASCII.

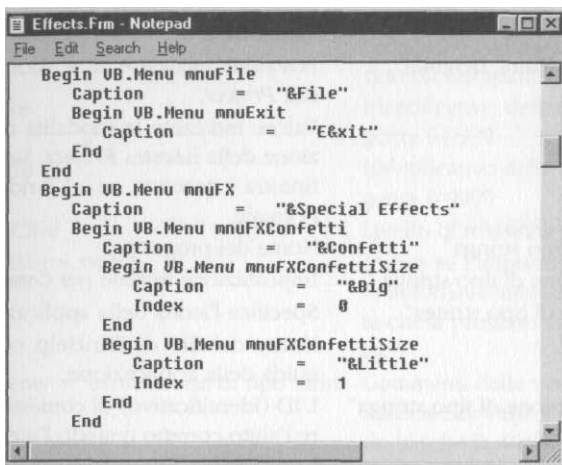
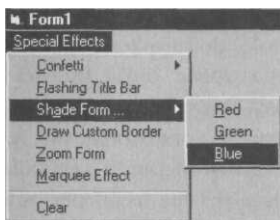



Figura 19.9
Potete risparmiare tempo copiando e incollando in ASCII strutture dimenu.

All'interno dei file di progetto

Anche i file di progetto di Visual Basic (.Vbp) sono salvati in file di testo ASCII. I file di progetto elencano i nomi dei file delle form, dei moduli, degli oggetti e dei riferimenti inclusi.

Se un progetto presenta problemi in fase di caricamento per la mancanza di suoi riferimenti (o indica file in posizione non corretta), potete editare manualmente il file . Vbp come testo informato ASCII consentendovi di rimediarefacilmente.

La Tabella 19.4 mostra le possibili parole chiave e valori che potete includere in un file di progetto .Vbp. Notate che non è necessario immettere il contenuto del file di Progetto .Vbp nell'ordine presentato nella Tabella 19.4.

Tabella 19.4 *File diprogetto (. Vbp) e loro contenuto*

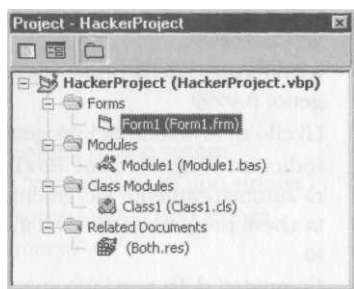
Parole chiave e valori	Significato
Type = "Tipo di progetto"	Tipo di progetto, per esempio, Exe o Control
Object = "percorso dell'oggetto"	Un controllo Active X o un oggetto inseribile. Vi è una voce separata per ogni controllo od oggetto.
References = (elenco dei riferimenti)	Indica le vostre impostazioni nella finestra di dialogo *References;* elenca ogni impostazione addizionale a quelle di default di Visual Basic .
ProjWinSize = top[, left[, widtht, height]]]	Posizione iniziale e dimensione della finestra *Project.*
ProjWinShow = show	Valore indicante la modalità di visualizzazione della finestra *Project.* Se show è 0, la finestra è nascosta, se è 1 è ridotta, se è 2 è normale.
Name = "espressione di tipo stringa"	Nome del progetto.
Command32 = "espressione di tipo stringa "	Impostazione iniziale per `Command`.
IconForm = "espressione di tipo stringa"	Specifica l'icona della applicazione.
HelpFile = "percorso"	Nome del file di WinHelp contenente la guida della applicazione.
HelpContextID = "espressione di tipo stringa"	L'ID (identificativo) di contesto per indicare l'aiuto corretto quando l'utente fa clic su ? nell'Object Browser dopo avere selezionato l'Object Library dell'applicazione.
ResFile32 = "nome di file	Il file di risorse associato al progetto
Title = Espressione di tipo stringa"	Il titolo dell'applicazione utilizzato sulla scheda *Make* della finestra di dialogo *Project Properties.*
CondComp = espressione[; espressione]...	Argomenti condizionali per la compilazione, separati da punto e virgola.
EXEName32 = "espressione di tipo stringa"	Il nome del file usato per indicare il .EXE nella scheda *Make* della finestra di dialogo *Project Properties.*
Path32 = "espressione di tipo stringa"	Le impostazioni di percorso indicate sulla scheda *Make* della finestra di dialogo *Project Properties.* Specifica dove è posto il file eseguibile dopo essere stato creato.
StartMode = switch	Riporta l'impostazione StartMode eseguita sulla scheda *Component* della finestra di dialogo *Project Properties.*
Description = "espressione di tipo stringa"	Descrizione del progetto. È visibile nell'Object Browser una volta selezionato il progetto.

Parole chiave e valori	Significato
rompatlbleEXE = percorso	Usato solo con i componenti ActiveX. Riporta il percorso di una precedente versione dell'eseguibile ed è il riferimento utilizzato per determinare quali sono i cambiamenti fatti che determinano l'incompatibilità nel controllo delle applicazioni utilizzando una versione meno recente di questo progetto.
VersionCompatlble = switch	Impostato da Visual Basic (non deve essere modificato manualmente). Indica se il progetto è compatibile con il file .EXE indicato da CompatlbleEXE.
MajorVer = Cifre	Identificativo dell'ultima versione del progetto; 0-9999.
MinorVer=Cifre	Identificativo della prima versione del progetto; 0-9999
RevisionVer=Cifre	Livello di revisione del progetto; 0-9999
AutoIncrementVer = switch	Indica se l'impostazione RevisionVer sarà automaticamente incrementata ogni volta che il progetto viene modificato e salvato.
VersionComments = "espressione di tipo stringa"	Commenti della versione specifica. L'informazione di versione è impostata utilizzando la scheda *Make* della finestra di dialogo *Project Properties*
VersionCompanyName =	Nome della società che ha sviluppato questa versione. L'informazione di versione è impostata utilizzando la scheda *Make* della finestra di dialogo *Project Properties.*
VersionFileDescription = "espressione di tipo stringa"	Indicazione del file di versione. L'informazione di versione è impostata utilizzando la scheda *Make* della finestra di dialogo *Project Properties.*
VersionLegalCopyright = "espressione di tipo stringa"	Informazioni di Copyright applicabili a questa versione. Impostate questa informazione utilizzando la scheda *Make* della finestra di dialogo *Project Properties.*
VersionLegalTrademarks = "espressione di tipo stringa"	Informazioni sui marchi commerciali utilizzati con questa versione. Impostate questa informazione utilizzando la scheda *Make* della finestra di dialogo *Project Properties.*
VersionProductName = "espressione di tipo stringa"	Nome del prodotto utilizzato in questa versione. Impostate questa informazione utilizzando la scheda *Make* della finestra di dialogo *Project Properties.*

Il file .Vpb include anche le impostazioni modificabiliper la compilazione nativa di codice per ogni progetto, come descritto nel Capitolo 16. Se anche venisse selezionata la modalità di compilazione in pseudocodice anziché quella nativa, ritroverete comunque inclusi iparametriper il codice nativo, insieme con la riga seguente (che significa "compilare in pseudocodice"):

```
CompilationType=-1
```

Per esempio, considerate un normale progetto avente un file di progetto chiamato HackerProject.Vbp. Esso contiene moduli, oggetti e riferimenti. Ciò è mostrato nel Project Explorer nella Figura 19.10. Il Listato 19.9 mostra il contenuto del file di progetto che corrisponde alla gerarchia mostrata nel Project Explorer:

Figuro 19.10 *L'HackerProject contiene differenti tipi di file.*

Listato 19.9 *Un file .Vbp d'esempio.*

```
Type=Exe
Reference=*\G{00020430-0000-0000-C000-
   000000000046}#2.0#0#C: \WINDOWS
   \SYSTEM\STDOLE2.TLB#OLE Automation
Object={6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.1#0; COMCTL32.OCX
Object={F6125AB1-8AB1-11CE-A77F-08002B2F4E98}#2.0#0; MSRDC20.OCX
Reference=*\G{56A868B0-0AD4-11CE-B03A-
   0020AF0BA770}#1.0#0#C:\WINDOWS\SYSTEM\quartz.dll#ActiveMovie
   control type library
Reference=*\G{8A41BBBC-ABF3-11CF-B8E7-0080C6026268}
   #1.0#0#C:\PROGRAM FILES\PLUS!\MICROSOFT
   INTERNERÀ-VRML\AVVIEW.oca#ActiveVRML
   Control Library
Reference=*\G{24807AE2-1BC8-11D0-B49B-
   00A0C911E8B6}#1.0#0#.. \..\VB\WINAPI\
   APILOAD.EXE#API Declaration Leader
Form=Form1.frm
Module=Module1; Module1.bas
Class=Class1; Class1.cls
ResFile32="..\Ch18\SourceCode\Both.res"
IconForm="Form1"
Startup="Form1"
HelpFile="HackerHelp"
Title="HackerProject"
```

```
Command32=""
Name="HackerProject"
HelpContextID="0"
Description="Hacker needs caffeine badly..."
CompatibleMode="0"
MajorVer=1
MinorVer=42
RevisionVer=0
AutoIncrementVer=1
ServerSupportFiles=0
VersionComments="This is just a demo"
VersionCompanyName="BearsOnWheels.com"
VersionFileDescription="Hacker's Text"
VersionLegalCopyright="None"
VersionLegalTrademarks="None"
VersionProductName="VB6 Secrets"
CompilationType=-1
OptimizationType=0
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
UnroundedFP=0
StartMode=0
Unattended=0
ThreadPerObject=0
MaxNumberOfThreads=1
```

Potreste aggiungere un progetto UserControl chiamato OurGroup.Vbp all'Hacker-Project, creando un gruppo di progetto di nome OurGroup.Vbg. La gerarchia di progetto creata è mostrata nel Project Explorer nella Figura 19.11.

Figura 19.11
Gruppi di progetto sono formati da più file di progetto.

In questo caso, il nuovo file di gruppo di progettto, OurGroup.Vbg, mostra i file di Progetto che contiene:

```
VBGROUP 5.0
StartupProject=HackerProject.vbp
Project=OurGroup.vbp
```

I file di gruppi di progetto, sia VB5 sia VB6, iniziano con VBGROUP 5.0.

Riepilogo

Potete utilizzare effetti speciali per rendere accattivante un'applicazione. Gli utenti apprezzano effetti visivi fuori dalla norma, a meno che non siano eccessivi. Gli effetti speciali possono rompere la monotonia di un lavoro noioso con qualcosa di divertente, attirando l'attenzione dell'utente o semplicemente rendendo una form visivamente più carino.
Le uova pasquali sono scritte inutili o vanitose che molti programmatori aggiungono alle loro applicazioni. Sebbene la maggior parte delle uova siano accessibili con un semplice click del mouse o particolari combinazioni di tasti, possono essere difficili da individuare.

- Avete imparato come creare e visualizzare effetti speciali
- Avete scoperto come creare coriandoli
- Ho spiegato come si usa la funzione QBColor
- Ho spiegato come utilizzare la funzione RGB.
- Ho trattato il metodo Line
- Avete imparato la funzione FlashWindow
- Avete imparato le proprietà Scale
- Avete imparato a sfumare un form.
- Avete anche imparato a disegnare i contorni di un form.
- Ho trattato i dettagli di un form che esplode.
- Avete imparato a recuperare un handle di contesto di dispositivo per lo schermo.
- Avete imparato a usare la funzione Rectangle per disegnare direttamente sullo schermo.
- Ho trattato come creare un effetto marquee
- Ho spiegato come usare il metodo Print.
- Avete imparato a creare testo tridimensionale.
- Avete scoperto come attivare un uovo di Pasqua.
- Avete imparato a far muovere l'uovo.
- Avete imparato a manipola form (.Frm), progetti (.Vbp) e gruppi di progetto (.Vbg) come file di testo ASCII.

CAPIRE ACTIVEX E OLE

- L'evoluzione di ActiveX
- Definizione di oggetto OLE e modello a oggetti ActiveX
- Visual Basic 6 e gli oggetti ActiveX
- Le MAPI (Messaging API)
- File composti e memoria strutturata
- I componenti ActiveX e il Registro

Negli ultimi anni ActiveX ha assunto una notevole importanza per gli sviluppatori a cui interessa scrivere applicazioni che utilizzino funzionalità di altri programmi, o, viceversa, creare oggetti che possano essere utilizzati all'interno di altre applicazioni. La tecnologia ActiveX è quella che fino a poco fa era nota con il nome di OLE. Questo capitolo presenta gli elementi necessari ad afferrare la continua evoluzione delle tecnologie ActiveX e OLE.

L'evoluzione di ActiveX

Le applicazioni Visual Basic 6 possono comprendere funzionalità ActiveX (cioè OLE) come il drag and drop. Il passo successivo per Visual Basic e ActiveX sono gli eseguibili ActiveX (i progetti ActiveX Exe) e le DLL ActiveX (i progetti ActiveX DLL), applicazioni server OLE controllabili da client OLE. I progetti dei documenti ActiveX (Exe e DLL) sono form, dotati di proprietà, eventi e metodi, visualizzabili m Internet Explorer. I controlli ActiveX sono componenti compilati (con estensione .OCX) utilizzabili da qualsiasi applicazione possa ospitare controlli OLE. Riassumendo, Visual Basic permette di creare:

Eseguibili standard che "parlano" OLE grazie all'implementazione di funzionalità come il drag and drop.

Applicazioni che fanno da client OLE per applicazioni server OLE ActiveX.

Server ActiveX controllabili da altre applicazioni.

- Documenti ActiveX, cioè Form visibili all'interno del browser di Internet Explorer o utilizzando il Raccoglitore di Office.
- Controlli ActiveX, veri e propri componenti condivisi.

Nel Capitolo 14 abbiamo visto cosa offra Visual Basic per la creazione di programmi orientati agli oggetti (tra l'altro, collezioni e moduli di classe). Ma Visual Basic ha un posto di primo piano in un panorama di oggetti ben più ampio di quello che mette a disposizione al suo interno. L'obiettivo reale della programmazione ad oggetti è sfruttare programmi già esistenti, vostri o scritti da altri: perché reinventare ogni volta la ruota della programmazione? Sotto questa ottica, il vero modello ad oggetti adottato da Visual Basic assomiglia più a quello di Windows che non a quello di molti linguaggi *object-orìented* di derivazione accademica.
Questa è la stessa considerazione da cui nasce ActiveX, che agisce in gran parte del sistema operativo Windows e in quasi tutte le attuali applicazioni ad esso destinate. ActiveX è un tentativo in continua evoluzione di plasmare un futuro orientato agli oggetti indipendente dai linguaggi di programmazione e dalle piattaforme. Per questo la possibilità di creare server e controlli ActiveX con Visual Basic è estremamente potente e importante.
La Parte V di questo libro inizia l'esplorazione del ruolo di Visual Basic in questo universo di oggetti, che si presenta sotto quattro facce:

- La facilità con cui si possono includere oggetti OLE in un progetto Visual Basic utilizzando il controllo OLE Container.
- La possibilità che i programmi Visual Basic, comportandosi da client, manipolino i metodi degli oggetti esposti da applicazioni server ActiveX, come quelle che costituiscono Microsoft Office.
- La possibilità di creare applicazioni Visual Basic che siano esse stesse server ActiveX, utilizzabili all'interno delle vostre applicazioni client o di quelle di altri.
- La possibilità, offerta dall'edizione Enterprise di Visual Basic, di gestire e creare server ActiveX remoti.

La creazione di controlli ActiveX con Visual Basic 6 è talmente stimolante che ha meritato una sezione a parte nel libro, la Parte VI. Inoltre, troverete informazioni sui documenti ActiveX nel Capitolo 28.

Che cos'è un oggetto OLE?

OLE, nell'accezione utilizzata in questo libro, indica lo standard per il collegamento e l'incorporamento di oggetti (Object Linking and Embedding) nella versione *2.x.* Tuttavia, la definizione "object linking and embedding" è diventata fuorviante: collegare e incorporare oggetti è solo una parte del gioco. Anche se la versione iniziale dello standard OLE (OLE 1) era stata ideata per consentire il collegamento e l'incorporamento di oggetti in documenti composti, l'obiettivo di OLE 2 è permettere e facilitare l'integrazione dei componenti e la scrittura dei programmi. Kraig Brockschmidt, esperto di OLE, definisce OLE in questo modo:

"OLE è un ambiente unificato di servizi orientati agli oggetti, in grado sia di adattare tali servizi sia di estendere l'architettura adottando servizi personalizzati, allo scopo di consentire una più stretta integrazione tra i componenti".

In termini meno astratti, OLE definisce uno standard coerente che permette agli oggetti, alle applicazioni e ai componenti ActiveX, di *comunicare* tra di loro allo scopo di utilizzare uno il codice degli altri: non è necessario che gli oggetti sappiano in anticipo con quali altri oggetti avranno a che fare, né che il loro codice sia scritto nello stesso linguaggio.

Le applicazioni ActiveX sono concettualmente distinte tra *server,* oggetti che mettono metodi e proprietà a disposizione degli altri, e *client,* applicazioni che utilizzano gli oggetti, i metodi e le proprietà esposti dai server (si noti che nulla vieta ad un'applicazione di essere contemporaneamente client e server). Alcuni tipi di server, per esempio i controlli ActiveX, possono far scattare eventi che vengono raccolti dal codice del client.

Esiste una terminologia alternativa che probabilmente può risultare più intuitiva: a volte ci si riferisce ai client OLE come controller, *e sembra giusto pensare che i client controllino i server OLE, che permettono per pane loro di essere controllati.*

Comunicazioni asincrone e sincrone

OLE non si limita alla comunicazione tra oggetti (come appena visto): la comunicazione è anche *sincrona,* avviene cioè in due direzioni (Figura 20.1). L'applicazione chiamante (client) emette una chiamata ed aspetta una risposta. L'applicazione ricevente (server) aspetta una chiamata, e quando la riceve invia una risposta all'applicazione chiamante che attende in linea.

Figura 20.1

Le conversazioni OLE 2 sono sincrone.

Comunicazione sincrona

Applicazione chiamante (client)
Emette una chiamata
Attende una risposta

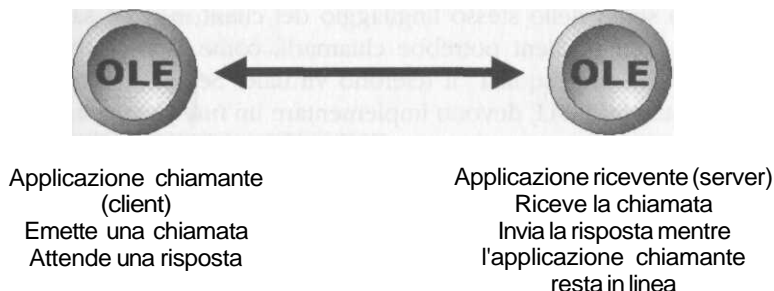
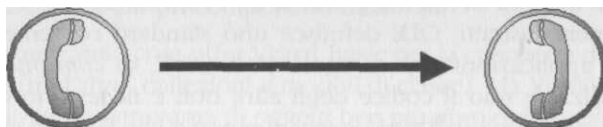
Applicazione ricevente (server)
Riceve la chiamata
Invia la risposta mentre l'applicazione chiamante resta in linea

Gli standard più datati per la comunicazione tra oggetti, come OLE 1 e DDE, utilizzavano una comunicazione asincrona. In questo caso l'applicazione chiamante non aspetta la risposta alla sua chiamata (Figura 20.2). La comunicazione asincrona è tipica dei radioamatori, o della posta elettronica su Internet: non potete essere sicuri che il vostro messaggio sia arrivato al destinatario fino a quando non ricevete una notifica esplicita.

Figura 20.2
La comunicazione asincrona è unidirezionale.

Comunicazione asincrona

Applicazione chiamante (client)
Emette una chiamata
Non aspetta la risposta

Applicazione ricevente (server)
Riceve la chiamata

Come sa bene chi ha scritto programmi che utilizzavano il DDE, la comunicazione asincrona tra oggetti può dare molti problemi: in mancanza di una notifica esplicita da parte dell'applicazione server è semplicemente impossibile sapere se la richiesta del client è stata soddisfatta. La comunicazione può scadere. L'elenco dei fattori che possono far fallire una comunicazione asincrona è alquanto esteso. Per molti motivi le comunicazioni asincrone tra oggetti sono meno affidabili e più complicate da programmare rispetto a quelle sincrone.
Grazie al protocollo di comunicazione sincrona di OLE 2, non dovete preoccuparvi di sapere se la vostra chiamata va a buon fine: le funzioni di comunicazione che chiamate non restituiscono il controllo finché il programma server non completa le proprie attività, e possono quindi restituire un valore che indica se l'operazione è andata a buon fine, e, in caso contrario, il motivo.

L'interfaccia OLE

Visto che uno dei cardini del modello OLE è che non è necessario né che gli oggetti server siano scritti nello stesso linguaggio dei client, né che sappiano in anticipo che tipo di oggetti client potrebbe chiamarli, come fanno server e client OLE a comunicare quando "squilla" il telefono virtuale? Semplice: gli oggetti che aderiscono allo standard OLE devono implementare un'*interfaccia* standard.
Gli oggetti OLE possono avere tutte le interfacce che si vuole, generalmente raggnippate per funzione. Una data interfaccia mostra una specie di inventario delle funzioni che contiene e fornisce il modo per eseguirle.
L'Object Browser utilizza l'interfaccia esposta dagli oggetti ActiveX per elencare i membri (proprietà, metodi ed eventi) del componente o dell'applicazione. I client Visual Basic possono utilizzare queste proprietà e metodi, e rispondere agli eventi, anche se i dettagli implementativi dell'interfaccia sono nascosti (a meno che non sia accessibile il codice sorgente, in Visual Basic o altri linguaggi). Per accedere ai membri dei server ActiveX, le vostre applicazioni client non devono far altro che utilizzare la nota sintassi *OggettoMetodo* o *Oggetto.Proprietà.* Gli eventi che possono essere scatenati da un oggetto come un controllo ActiveX vengono elencati nella struttura per la gestione degli eventi della finestra del codice del client: potete aggiungere codice per trattare gli eventi scatenati da un componente ActiveX.

Definizione di oggetto ActiveX

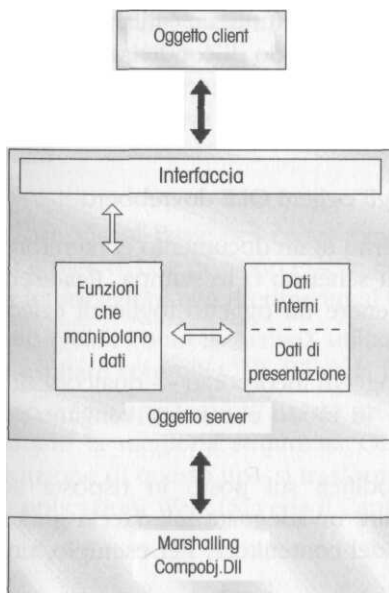
Un oggetto si definisce ActiveX se rispetta il modello Component Object Model (COM), definito originariamente dalla Microsoft e successivamente rilasciato ad un gruppo pubblico del settore, l'Active Group. A quanto si legge sul loro sito Web, l'Active Group sarebbe "un'associazione di settore aperta per la promozione dell'utilizzo delle tecnologie ActiveX".

Per avere più informazioni sull'Active Group visitate il loro sito: `http://www.activex.org`.

Un oggetto conforme allo standard COM ha le seguenti caratteristiche:

- è implementato in codice binario, quindi può essere scritto in un linguaggio sorgente qualsiasi.
- è incapsulato in un file eseguibile (normalmente .Exe per le applicazioni e .Ocx per i controlli) o in una libreria a collegamento dinamico (.Dll).
- contiene due tipologie di dati: *dati di presentazione,* necessari per la visualizzazione su schermo o la stampa, e *dati interni.* Potete pensare i due tipi di dati come proprietà private dell'oggetto.
- contiene anche funzioni per la manipolazione di questi dati.
- fornisce agli altri oggetti un'interfaccia standard (vista prima) con cui comunicare con esso.
- partecipa allo smistamento *(marshaling),* il processo di trasferire gli argomenti e i valori di ritorno delle funzioni tra processi e macchine. Lo smistamento è gestito da routine interne al file di sistema Compobj.Dll.

La Figura 20.3 rappresenta la struttura generica di un oggetto OLE 2.

Figura 20.3 *La struttura interna di un oggetto server OLE prevede un'interfaccia e una serie di funzioni per la manipolazione dei dati.*

Che cosa fa un oggetto ActiveX

In realtà, quello che un'applicazione ActiveX fa in qualità di server OLE è sostanzialmente aspettare, soprattutto i server OLE, che il più delle volte non fanno nulla fino al momento in cui sono chiamati. Naturalmente, non pochi oggetti mostrano entrambe le facce: restano in attesa di una chiamata come server, ma nel frattempo sono impegnati, e magari stanno chiamando altri oggetti server come client. Per esempio, Word potrebbe essere chiamato come server da un oggetto client esterno e nel frattempo essere occupato nell'aggiornamento di un foglio elettronico incorporato.
In generale, un oggetto OLE dovrebbe supportare una serie di protocolli e fornire alcuni servizi:

- dovrebbe fornire un'interfaccia per i suoi comandi interni (cioè automatizzare), in modo che gli altri oggetti possano far compiere al server determinate operazioni sui suoi dati: da qui la frase "OLE automation server" (server per l'automazione OLE). Per esempio, un oggetto Excel fornisce un metodo che permette ad un client esterno di fargli caricare un foglio di calcolo.
- dovrebbe supportare il drag and drop. All'interno della finestra dell'oggetto, dovrebbe reagire correttamente agli oggetti che vengono trascinati e rilasciati su di esso con il mouse.
- deve supportare lo Uniform Data Transfer (UDT), un meccanismo di gestione degli scambi tra applicazioni di strutture dati formattate. I trasferimenti UDT avvengono comunicando le informazioni sui puntatori, anziché i dati veri e propri, in modo che non sia necessario leggere in memoria grandi quantità di dati.
- gli oggetti OLE dovrebbero partecipare all'architettura definita nello standard OLE come file *composti* a memoria strutturata (utilizzando un servizio OLE). I file composti definiscono un metodo di condivisione dei contenuti di un file tra i componenti, utilizzando un meccanismo che può essere visto come "un file System all'interno di un file System" (come diremo tra poco).

Nell'ambito dei documenti composti, gli oggetti OLE dovrebbero:

- incorporarsi correttamente all'interno di un documento contenitore e riprodurre fedelmente i propri dati su schermo o in stampa. Per esempio, un documento Word potrebbe contenere un oggetto foglio di calcolo Excel incorporato.
- conservare i collegamenti tra gli oggetti incorporati — qualcosa di simile ai collegamenti DDE automatici — in modo che i dati vengano aggiornati automaticamente.
- prevedere l'attivazione e la modifica sul posto in risposta ad azioni dell'utente. Questo significa fornire un'adeguata interfaccia grafica per la modifica all'interno del contesto del contenitore. Per esempio, un oggetto

CorelDraw incorporato in un documento Word può essere modificato utilizzando gli strumenti Corel.

Gli oggetti OLE dispongono al loro interno di un oggetto detto *moniker,* che incapsula un puntatore ad un oggetto e i meccanismi per ricreare tale puntatore se necessario. Nella terminologia DDE, il puntatore è un percorso all'oggetto collegato, accompagnato da un metodo per localizzarlo nell'eventualità in cui il percorso assoluto non fosse più valido.

Visual Basic 6 e ActiveX

In generale, Visual Basic 6 tratta gli oggetti ActiveX in vari modi:

- Si possono installare controlli ActiveX nella toolbox di Visual Basic, ed in seguito incorporarli in un forni. Le proprietà del controllo possono essere impostate nella finestra *Properties* o dal codice, i suoi metodi possono essere chiamati nel codice, ed è possibile utilizzare codice Visual Basic nelle routine di gestione degli eventi pubblicati.
- Una volta che un oggetto ActiveX è stato istanziato, utilizzando l'istruzione Dim e la parola chiave New o il comando Set, si può operare all'interno del codice con i membri esposti dal server ActiveX. Per richiamare le proprietà e i metodi di un'istanza dell'oggetto si usano la variabile che lo rappresenta e l'operatore punto: questa tecnica può essere utilizzata, per esempio, per includere e manipolare fogli di calcolo Excel e documenti Word per Windows in applicazioni Visual Basic.
- Si può usare il controllo OLE — la cui icona nella Toolbox è riprodotta nella Figura 20.4 — come contenitore per incorporare oggetti in un forni. Il controllo, le sue proprietà e i suoi metodi possono essere considerati come un'interfaccia per oggetti incorporati e collegati. Vedremo il controllo OLE in dettaglio nel Capitolo 21.

Figura 20.4
L'icona delcontrollo OLE.

Si può usare Visual Basic per creare oggetti server OLE ActiveX dotati di proprietà e metodi che possono essere richiamati da oggetti client VB. Per maggiori informazioni fate riferimento al Capitolo 23.

Si possono creare controlli ActiveX utilizzabili da qualsiasi applicazione in grado di ospitare controlli OLE. (Si veda la Parte VI.)

Si possono creare applicazioni basate sui documenti ActiveX, cioè in grado di supportare la memoria strutturata OLE. Sostanzialmente, sviluppando un'applicazione di questo tipo si trasformano i forni Visual Basic in vere e proprie applicazioni Web. (Si veda il Capitolo 28.)

Visual Basic e il drag and drop

OLE implementa il drag and drop per semplificare il trasferimento dei dati ed altre informazioni da un oggetto detto "sorgente" ad un altro detto "destinazione"- l'applicazione sorgente costruisce un oggetto contenente i dati, che diventa l'oggetto dell'operazione di trascinamento; un'applicazione, quando riceve la notifica del rilascio dell'oggetto, esegue tutte le operazioni necessarie per ricevere i dati
I controlli che è possibile trascinare — praticamente quasi tutti, tranne i controlli Line, Menu, Shape, Timer e CommonDialog — dispongono di due proprietà impostabili:

- DragMode stabilisce se il trascinamento sia automatico o manuale.
- DragIcon imposta l'icona da utilizzare durante l'operazione di trascinamento.

Nel caso di trascinamento automatico, Visual Basic gestisce l'operazione riconoscendo l'inizio del trascinamento, modificando il cursore di conseguenza e notificando alla destinazione il rilascio. Nel caso invece del trascinamento manuale, la gestione è interamente a carico vostro: l'applicazione deve avviare il trascinamento chiamando il metodo .Drag, trattare il trascinamento nel proprio codice di gestione degli eventi, modificare il cursore e avvertire la destinazione. Il Capitolo 21 presenta alcuni esempi di trascinamento sia automatico che manuale.
II metodo Drag avvia, termina o annulla un'operazione di trascinamento per qualsiasi controllo che preveda il drag and drop. La sintassi del metodo è

oggetto.drag *azióne*

La Tabella 20.1 elenca i possibili valori del parametro *azione.*

Tabella 20.1 *Possibili valori delparametro azione del metodo Drag.*

Costante	Valore	Descrizione
vbCancel	0	Annulla l'operazione di trascinamento
VbBeginDrag	i	Inizia il trascinamento dell'oggetto
VbEndDrag	2	Termina il trascinamento dell'oggetto e lo rilascia

Per iniziare un trascinamento automatico non è necessario usare il metodo Drag: è uno dei motivi per utilizzare la gestione automatica del drag and drop, impostando DragMode ad automatic. In certi casi può anche capitare di voler iniziare un'operazione di trascinamento chiamando il metodo Drag anche se la sua gestione è automatica.

Fino alla versione 4 di Visual Basic, il metodo Drag *era asincrono, nel senso che tutte le istruzioni che lo seguivano venivano eseguite anche se Vazione di trascinamento non era stata completata. In Visual Basic 6, il metodo* Drag è *sincrono, e l'azione di trascinamento deve essere terminata prima che le istruzioni che lo seguono vengano eseguite.*

routine di gestione degli eventi che rispondono alle operazioni di trascinamento sono due:

DragDrop viene lanciato al completamento di un'operazione di drag and drop, cioè quando si trascina un controllo e si rilascia il pulsante del mouse al di sopra di un form o di un altro controllo, o si chiama il metodo Drag passando vbEndDrag come parametro per l'azione. Il codice scritto per l'evento DragDrop controlla cosa succede al termine del trascinamento.

DragOver viene lanciato, più volte, durante l'operazione di drag and drop. Per controllare quando il puntatore del mouse entra, esce o resta fermo nell'area di una destinazione valida si può usare il parametro state di questo evento. La posizione del puntatore del mouse determina l'oggetto che riceve questo evento.

Si può usare DragOver per stabilire che cosa succede tra l'inizio del trascinamento e il rilascio del controllo sulla destinazione. Per esempio, si potrebbe confermare all'utente che un oggetto è una destinazione valida evidenziandolo (impostandone la proprietà BackColor o ForeColor) o visualizzando un particolare puntatore di trascinamento (impostando la proprietà DragIcon o MousePointer dal codice). *State è* un valore intero che rappresenta lo stato, relativamente alla destinazione, del controllo che si sta trascinando; può assumere tre valori, descritti nella Tabella 20.2.

Tabella 20.2 *Possibili valori delparametro State dell'evento DragOver.*

Valore	Significato
0	Enter (Il controllo sorgente sta entrando nell'area del target.)
1	Leave (Il controllo sorgente sta uscendo dall'area del target.)
2	Over (Il controllo sorgente si sta spostando all'interno dell'area del target.)

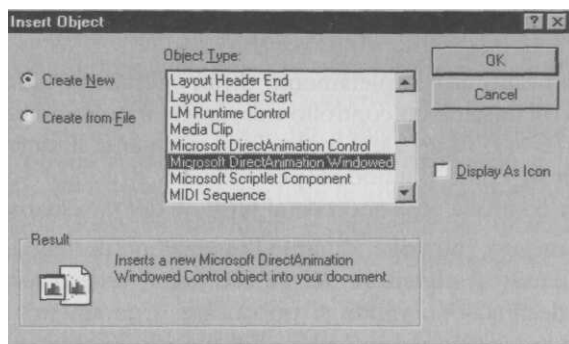
La procedura più semplice per impostare un'operazione di drag and drop comporta due soli passi:

1. Impostare le proprietà DragMode dei controlli che saranno sorgenti e destinazione dell'operazione ad automatic.
2. Aggiungere il codice per la corretta gestione degli eventi DragOver e DragDrop per i target.

Nel Capitolo 21 approfondiremo le tecniche per la programmazione del drag and drop.

Visual Basic e i contenitori

Per aggiungere un contenitore di oggetti incorporati a un form Visual Basic, basta inserire su un form un contenitore OLE utilizzando il controllo OLE e impostarne la Proprietà OLETypeAllowed a 1 -Embedded. In questo modo il controllo OLE viene impostato come contenitore di oggetti incorporati: potete selezionare l'oggetto che diventerà il contenuto del contenitore utilizzando la voce *Insert Object* del menu Pop-up del controllo OLE (Figura 20.5).

Figura 20.5 *Per incorporare oggetti in un form Visual Basic si usa il controllo Container.*

Gli oggetti collegati sono del tutto simili a quelli incorporati, con la differenza che il contenitore collegato dispone di un *moniker* che permette all'oggetto contenuto di localizzare la fonte a cui è collegato. Per configurare un controllo OLE in modo che possa contenere un oggetto collegato se ne imposta la proprietà OLETypeAllowed a 0-Linked. In seguito, per collegare gli oggetti al contenitore si usa la voce *Paste Special* del menu pop-up del controllo OLE.

Un *server incorporato (embedded server) è* un server in grado di creare un oggetto all'interno del contenitore. Per esempio, se all'interno di un controllo OLE contenitore si mette un documento Word generico, quando si fa doppio clic sul controllo in fase di esecuzione viene lanciato Word, permettendo all'utente di creare il particolare oggetto che si desidera incorporare, come si vede nella Figura 20.6. È la cosiddetta *in-place activation:* si dice che l'attivazione avviene "sul posto" o "in loco" quando non è necessario abbandonare l'applicazione del contenitore per effettuare l'azione predefinita per l'oggetto quando viene attivato.

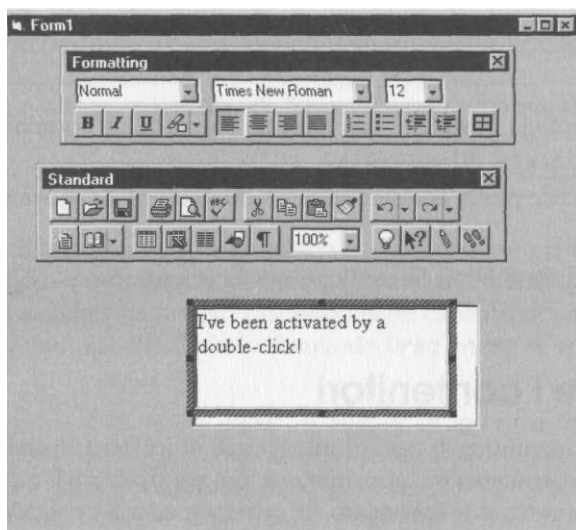
Figura 20.6 *Facendo doppio clic su un documento Word generico all'interno di un controllo OLE si attiva una piccola versione dell'ambiente di modifica di Word.*

Per ogni oggetto è definita un'azione di default, che viene eseguita quando si fa doppio clic sul controllo se la proprietà AutoActivate è impostata a vbOLEActivateDoubleclick (un valore costante pari a 2). Il più delle volte questo è tutto quello *che* serve per aprire l'ambiente di editing dell'oggetto e poterlo modificare; tuttavia esistono anche altre azioni possibili: per esempio, quando viene attivato, un oggetto collegato potrebbe semplicemente aggiornarsi.
Per personalizzare la risposta dell'oggetto al doppio clic scrivendo un gestore specifico per l'evento, è prima necessario impostare la proprietà AutoActivate del controllo a vbOLEActivateManual (pari a 0). Per aprire un oggetto per svolgere una certa operazione (per esempio la modifica) si usa il metodo DoVerb, specificando nel parametro *verb* l'operazione desiderata. Nonostante ogni oggetto preveda un proprio insieme di operazioni, in Tabella 20.3 è riportato un elenco di operazioni standard che ogni oggetto OLE dovrebbe prevedere.

Tabella 20.3 *Argomenti standardper il metodo DoVerb.*

Costante	**Valore**	**Descrizione**
VbOLEPrimary	0	L'azione predefinita per l'oggetto.
VbOLEShow	-1	Attiva l'oggetto per la modifica. Se l'applicazione che ha creato l'oggetto prevede l'attivazione "in-piace", l'oggetto viene attivato all'interno del controllo OLE contenitore.
VbOLEOpen	-2	Apre l'oggetto in una finestra separata. Se l'applicazione che ha creato l'oggetto prevede l'attivazione "in-place", l'oggetto viene attivato all'interno di una propria finestra.
VbOLEHide	-3	Nasconde l'applicazione che ha creato l'oggetto nel caso di oggetti incorporati.
VbOLEUIActivate	-4	Se l'oggetto prevede l'attivazione sul posto, attiva l'oggetto visualizzando eventuali strumenti dell'interfaccia utente. In caso contrario, l'oggetto non viene attivato e viene generato un errore.
VbOLEInPlaceActivate	-5	Se l'utente sposta il focus sul controllo OLE contenitore, crea una finestra per l'oggetto e lo prepara alla modifica. Se l'oggetto non prevede l'attivazione in risposta a un clic singolo del mouse si verifica un errore.
VbOLEDiscardUndoState	-6	Utilizzato per un oggetto attivato per la modifica per azzerare la storia delle modifiche annullabili dall'applicazione dell'oggetto.

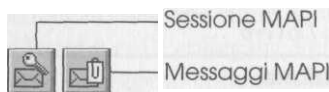
MAPI

A questo punto, le MAPI (Messaging API) potrebbero sembrare una divagazione. Quello che hanno in comune con OLE è la comunicazione tra oggetti, una comunicazione standardizzata. Le MAPI sono l'architettura ideata da Microsoft per connettere le applicazioni ad un'ampia varietà di servizi di messaggistica.

Da un lato, la disponibilità di semplici funzionalità di posta, per esempio un elemento che nel menu *File* permetta di inviare posta, è un requisito della Microsoft per il rilascio della conformità a Windows. Dall'altro, a seconda del tipo di applicazione che si realizza, l'utente potrebbe aspettarsi o pretendere una gestione sofisticata dei messaggi. Per realizzare le funzionalità MAPI si possono usare i controlli OLE MAPI (msmapi.ocx) o chiamare direttamente le Messaging API.

Uso dei controlli MAPI

Se non li trovate già nella Toolbox (Figura 20.7), potete aggiungere i controlli MAPI dalla finestra di dialogo *Components* accessibile dal menu *Project* di Visual Basic, selezionando Microsoft MAPI Controls 6.0 (msmapi32.ocx).

Figura 20.7

II controllo MAPI Session permette di aprire una sessione di messaging in funzione delle proprietà impostate nella finestra *Properties* o direttamente nel codice. Il controllo prevede due metodi: SignOn e SignOff.
Il metodo SignOn apre la finestra di *Logon* per l'utente dell'account specificato dalle proprietà UserName e Password, e restituisce un handle per il sottosistema di messaggistica, che viene conservato nella proprietà SessionID.
Per esempio, il codice phe segue attiva la finestra di *Logon* per l'utente definito dal profilo "MS Exchange Settings 1" (sul mio sistema questo corrisponde a Jean-Lue Picard, con password "MakeItSo"):

```
MAPISession1.UserName = "MS Exchange Settings 1"
MAPISession1.SignOn
```

Questa semplice applicazione diposta è disponibile sul CD-ROM come Mapi. Vbp.

Il passo successivo è la configurazione del controllo Messaggi MAPI mediantel'handle della sessione fornito dal controllo Sessione MAPI:

```
MAPIMessages1.SessionID = MAPISession1.SessionID
```

Impostando a -1 la proprietà MsgIndex si indica che si sta componendo un messaggio in uscita:

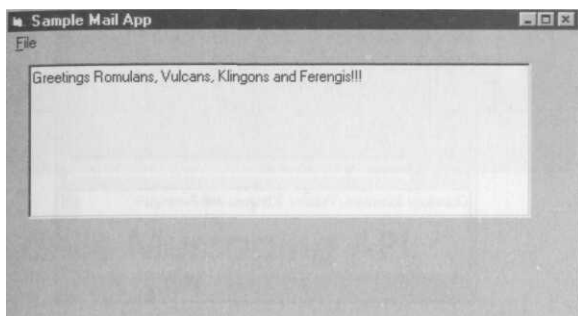
```
MAPIMessages1.MsgIndex = -1
```

Probabilmente, in un'applicazione reale, il testo del titolo del messaggio sarebbe prelevato da una casella di testo compilata dall'utente, e non impostato direttamente nel codice:

```
MAPIMessages1.MsgSubject = "Welcome to the Federation!"
```

Ti testo del messaggio, invece, viene letto da un controllo RichTextBox (Figura 20.8):

```
MAPIMessages1.MsgNoteText = RichTextBox1.Text
```

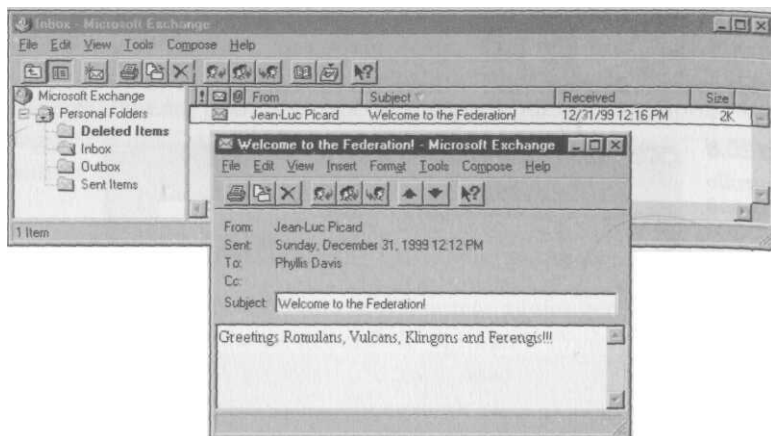
Figura 20.8
Un controllo RichTextBox puo essere utilizzato per permettere agli utenti di scrivere il testo dei messaggi di posta.

Infine, si visualizza la rubrica indirizzi dell'utente, si chiama il metodo ResolveName per verificare che il nome del destinatario corrisponda (è possibile intercettare l'errore se non si trova una corrispondenza), si invia il messaggio, e si chiude la sessione MAPI:

```
MAPIMessages1.Show
MAPIMessages1.ResolveName
MAPIMessages1.Send
MAPISession1 .SignOff
```

Ecco la procedura completa per un semplice invio MAPI:

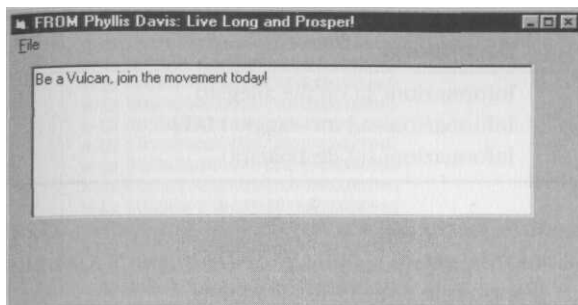
```
Private Sub mnuMail_Click()
    MAPISession1.UserName = "MS Exchange Settings 1"
    MAPISession1.SignOn
    MAPIMessages1.SessionID=MAPISession1.SessionID
    MAPIMessages1.MsgIndex=-1
    MAPIMessages1.MsgSubject = "Welcome to the Federation!"
    MAPIMessages1.MsgNoteText = RichTextBox1.Text
    MAPIMessages1.Show
    MAPIMessages1.ResolveName
    MAPIMessages1.Send
    MAPISession1.SignOff
End Sub
```

Come si può vedere nelle Figure 20.8 e 20.9, questo semplice esempio funziona davvero.

Figura 20.9
Il messaggio di posta generato da un'applicazione Visual Basic si presenta esattamente come gli altri al destinatario che lo apre.

Il procedimento per la ricezione della posta mediante i controlli MAPI è perfettamente analogo a quello appena visto. Questo è il codice necessario per leggere un messaggio nel controllo RichTextBox, mettendo il nome del mittente e l'argomento nel titolo di Form1:

```
Private Sub mnuGet_Click()
    MAPISession1.UserName = "MS Exchange Settings 1"
    MAPISession1.SignOn
    MAPIMessages1.SessionID = MAPISession1.SessionID
    MAPIMessages1.FetchUnreadOnly = True
    MAPIMessages1.FetchMsgType = ""
    MAPIMessages1.Fetch
    Form1.Caption = "FROM " + MAPIMessages1.MsgOrigDisplayName + _
        ": " + MAPIMessages1.MsgSubject
    RichTextBox1.Text = MAPIMessages1.MsgNoteText
    MAPISession1.SignOff
End Sub
```

Le proprietà del controllo MAPIMessages permettono di impostare varie opzioni. Vengono prelevati soltanto i messaggi non letti. La proprietà FetchMsgType stabilisce il tipo dei messaggi che vengono prelevati; una stringa nulla, il valore predefinito, indica i messaggi interpersonali.
Una volta lanciato il metodo Fetch, la proprietà MsgOrigDisplayName conterrà il nome del mittente, MsgSubject l'argomento del messaggio, e MsgNoteText il testo.
Come si vede dal codice (Figura 20.10), permettere alla vostra applicazione di ricevere messaggi è facile quanto consentirle di spedirli. Sappiate, comunque, che si tratta di esempi molto scarni — per esempio, non è gestito alcun tipo di errore, neanche il più comune come la pressione del pulsante *Cancel* nelle finestre di dialogo di Exchange aperte dai controlli.

Figura 20.10 Ricevere posta utilizzando i controlli MAPI è facile.

Funzioni delle Messaging API

Volendo, i controlli MAPI non sono realmente necessari: l'accesso alle funzioni MAPI (raccolte nel file Mapi32.Dll) è piuttosto semplice. La Tabella 20.4 elenca alcune semplici funzioni MAPI disponibili in Visual Basic, e la Tabella 20.5 mostra i relativi tipi.

Tabella 20.4 *Semplici funzioni MAPI.*

Funzione	Descrizione
MAPILogon	Apre una sessione con il sistema di messaggistica
MAPIFindNext	Restituisce l'ID del successivo (o primo) messaggio del tipo specificato
MAPIReadMail	Legge un messaggio di posta
MAPISaveMail	Salva un messaggio di posta
MAPIDeleteMail	Elimina un messaggio di posta
MAPISendMail	Invia un messaggio di posta, con una gestione più flessibile della generazione del messaggio rispetto a MAPISendDocuments
MAPISendDocuments	Invia un messaggio di posta standard utilizzando una finestra di dialogo
MAPIAddress	Imposta il destinatario di un messaggio di posta
MAPIResolveName	Visualizza una finestra di dialogo per la risoluzione di nomi di destinatari ambigui
MAPIDetails	Visualizza la finestra dei dettagli di un destinatario
MAPILogoff	Chiude una sessione con il sistema di messaggistica

Tabella 20.5 *i tipi MAPI.*

Tipo	**Descrizione**
MapiFile	Informazioni su un file allegato
MapiMessage	Informazioni sul messaggio MAPI
MapiRecip	Informazioni sul destinatario

Se desiderate approfondire questo argomento, la fonte migliore sulla dichiarazione e l'uso di queste funzioni è la sezione dell'*Office Developer's Kit* intitolata "Simple MAPI for Visual Basic", disponibile sul CD-ROM MSDN.

File composti e memoria strutturata

Quando si incorporano oggetti di un tipo in oggetti contenitore di un tipo differente, come nel caso di OLE, è necessario un meccanismo di memorizzazione che tenga traccia delle diverse tipologie di informazioni conservate e delle loro posizioni.

I file composti (compound file), un'implementazione del concetto di memoria strutturata *(structured Storage),* organizzano al loro intero le informazioni in *Storage,* analoghi alle strutture delle directory, e *stream,* analoghi ai file. Per salvare gli oggetti si sfrutta la capacità dell'applicazione incorporata di operare con file composti.

Inoltre, i documenti ActiveXforniscono un modo per utilizzare le tecniche per la memorizzazione strutturata OLE all'interno delle proprie applicazioni (maggiori informazioni nel Capitolo 28).

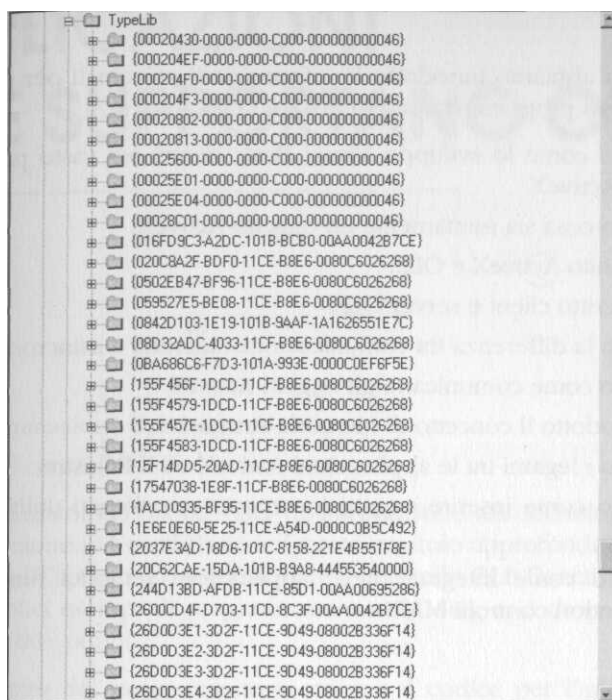
Le applicazioni ActiveX e il Registry

Le informazioni sugli oggetti OLE sono contenute nel Registro nella gerarchla HKEY_CLASSES_ROOT: le definizioni identificano le applicazioni per gli oggetti, la posizione dei relativi dati, ed un codice numerico univoco per ogni classe di oggetti utilizzata nell'applicazione (CLSID).

Per quanto fondamentali per il corretto funzionamento delle applicazioni OLE, queste definizioni sono piuttosto oscure (Figura 20.11), e probabilmente modificarle a mano non vi porterà molto lontano.

Fortunatamente, la maggior parte della registrazione delle applicazioni OLE VB non spetta a voi: Visual Basic si occupa automaticamente della registrazione temporanea delle applicazioni server che girano nell'IDE. Inoltre, i riferimenti ad applicazioni e controlli ActiveX nel codice utilizzano i nomi, e non i CLSID.

Un'applicazione ActiveX compilata viene registrata nel sistema la prima volta che viene eseguita. Inoltre, se si crea un programma di installazione utilizzando il *Setup Wizard* (si veda il Capitolo 35), la procedura generata registrerà automaticamente l'applicazione server OLE sul sistema di destinazione.

Figura 20.11 *Le definizioni di Class ID (CLSID) nel Registro non sono di facile interpretazione.*

L'utility Regsvr32, disponibile nella cartella Tools\REGISTRATIONUTILITIES del CD-ROM di VB6, permette di inserire ed eliminare la registrazione dei server. Il parametro della riga di comando /u *indica che la registrazione del server va rimossa. Per esempio, per registrare il server MyInProcessServer.Dll:*

```
C:\Vb\RegSvr32  MyInProcessServer.Dll
```

Per rimuovere la registrazione dello stesso componente ActiveX:

```
C:\Vb\RegSvr32  /u  MyInProcessServer.Dll
```

Anche se normalmente nei vostri programmi utilizzerete il nome dell'oggetto ActiveX, dovrete comprendere i CLSID per alcuni utilizzi, come la distribuzione di controlli ActiveX sul Web. Per maggiori informazioni sui CLSID ed il Registro, fate riferimento al Capitolo *9,* invece per la distribuzione sul Web dei controlli ActiveX scritti in Visual Basic fate riferimento al Capitolo 28.

Riepilogo

In questo Capitolo abbiamo introdotto le informazioni principali per poter approfondire l'esame della programmazione ActiveX e OLE.

- Abbiamo visto come lo sviluppo Visual Basic rientri nel vasto panorama degli oggetti ActiveX.
- Abbiamo visto cosa sia esattamente un oggetto ActiveX.
- Abbiamo definito ActiveX e OLE.
- Abbiamo descritto client e server OLE.
- Abbiamo visto la differenza tra comunicazione sincrona e asincrona.
- Abbiamo visto come comunicano gli oggetti ActiveX.
- Abbiamo introdotto il concetto di memoria strutturata.
- Abbiamo visto i legami tra le applicazioni ActiveX ed il Registro.
- Abbiamo visto come inserire oggetti incorporati o collegati utilizzando il controllo OLE.
- Abbiamo visto come integrare nelle applicazioni semplici funzionalità MAPI utilizzando i controlli MAPI.

APPLICAZIONI CHE SUPPORTANO OLE

- Programmazione del drag and drop
- Uso del controllo OLE

Nell'ultimo capitolo abbiamo visto un'introduzione alla tecnologia OLE e alla sua implementazione in Visual Basic 6. Questo capitolo approfondisce alcuni dei punti presentati nel Capitolo 20, in particolare il controllo OLE Container. Inizieremo con alcuni semplici esempi di programmazione sul drag and drop. È importante che siano chiari due punti:

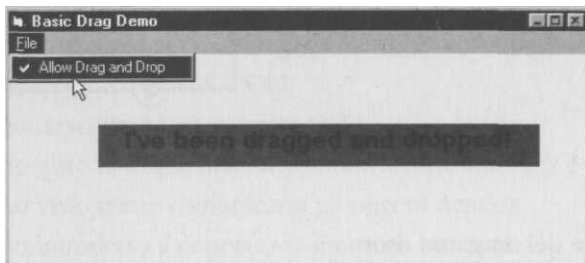
- È compito del programmatore scrivere il codice per l'implementazione della maggior parte di un'operazione di drag and drop.
- Sostanzialmente la meccanica di tale implementazione è sempre la stessa, indipendentemente dal ricorso o meno a OLE, almeno fino all'azione che termina l'operazione di drag and drop.

Esercizi di riscaldamento per il drag and drop

Il primo approccio alla programmazione del drag and drop che vi propongo (disponibile sul CD-ROM come Dragl.Vbp) dimostra come trascinare e rilasciare un controllo etichetta su un form. Nella finestra Properties, *ho impostato come didascalia iniziale per l'etichetta (di nome* lblDragJ *il valore "Drag me!". La proprietà* BackColor *di* lblDrag *è, per ora, impostata sul valore predefinito* vbButtonFace *(una costante pari a -2147483633). Se impostate la proprietà* DragMode *di* lblDrag *a* 1-Automatic, *quando lanciate il progetto potete trascinare l'etichetta a spasso per il form, ma al momento non succede molto altro.*

Facendo un piccolo passo avanti, potremmo impostare la proprietà DragIcon di lblDrag (sia dal codice che dalla finestra *Properties)* in modo che la forma del cursore cambi durante il trascinamento dell'etichetta. Nel progetto di esempio ho impostato come icona di trascinamento quella della nuvola con la pioggia disponi-

bile come Rain.Ico nella sottodirectory Elements della cartella di VB contenente gli esempi di icone — piuttosto indicativa del drag and drop, direi.
Adesso facciamo sul serio! Se aggiungiamo un menu al form del progetto, possiamo facilmente scrivere del codice che permetta all'utente di attivare e disattivare il trascinamento (vedi Figura 21.1).

Figura 21.1 *Gli eventi DragOver e DragDrop permettono di implementare facilmente operazioni di drag and drop.*

Perché il tutto possa funzionare, la proprietà DragMode di lblDrag deve essere impostata al valore predefinito 0-Manual, cioè "non trascinare a meno che non venga chiamato esplicitamente il metodo Drag". Ecco il codice per la procedura di gestione del menu:

```
Private Sub mnuDD_Click()
   If Not mnuDD.Checked Then
      lblDrag.DragMode = 1  'Automatico
   Else
      lblDrag.DragMode = 0  'Manuale
   End If
   mnuDD.Checked = Not mnuDD.Checked
End Sub
```

Potremmo anche far fare qualcosa all'etichetta durante il trascinamento: cambiare didascalia, colore e posizione. Questo codice va nell'evento DragOver della destinazione, nel nostro caso il forni:

```
Private Sub Form_DragOver(Source As Control, X As Single, _
Y As Single, State As Integer)
   Source.Caption = "Mi trascinano!"
   Source.BackColor = vbRed
   Source.Left = X
   Source.Top = Y
End Sub
```

Se ora provate a lanciare il programma, vedrete che tutto funziona: nel codice, S*ource* si riferisce al controllo etichetta, di cui vengono modificati la didascalia, il colore (che diventa rosso), e la posizione (che segue gli spostamenti del mouse).

Resta spazio per dei miglioramenti: cosi com'è scritto, il codice permette il trascinamento dell'etichetta fin quasi fuori il form sul lato destro e in basso. Possiamo aggiungere del codice per controllare e correggere questo comportamento. Il Listato 21.1 contiene la procedura perfezionata.

Listato 21.1 *Uso dell 'evento DragOverper spostare un controllo sul form.*

```
Private Sub Form_DragOver(Source As Control, X As Single, _
  Y As Single, State As Integer)
  Source.Caption = "Mi trascinano!"
  Source.BackColor = vbRed
  If X > Me.ScaleWidth - Source.Width Then
     X = Me.ScaleWidth - Source.Width
  End If
  Source.Left = X
  If Y > Me.ScaleHeight - Source.Height Then
    Y = Me.ScaleHeight - Source.Height
  End If
  Source.Top = Y
End Sub
```

Manca solo il codice per chiudere il drag and drop: in questo caso ci limiteremo a cambiare la didascalia e il colore dell'etichetta un'ultima volta, poiché i suoi parametri di posizionamento sono già stati impostati nell'ultima chiamata all'evento DragOver. Il codice per il rilascio va nell'evento DragDrop di lblDrag (non in quello del forni):

```
Private Sub lblDrag_DragDrop(Source As Control, X As Single, Y As Single)
  Source.Caption = "Sono stata trascinata e rilasciata!"
  Source.BackColor = vbBlue
End Sub
```

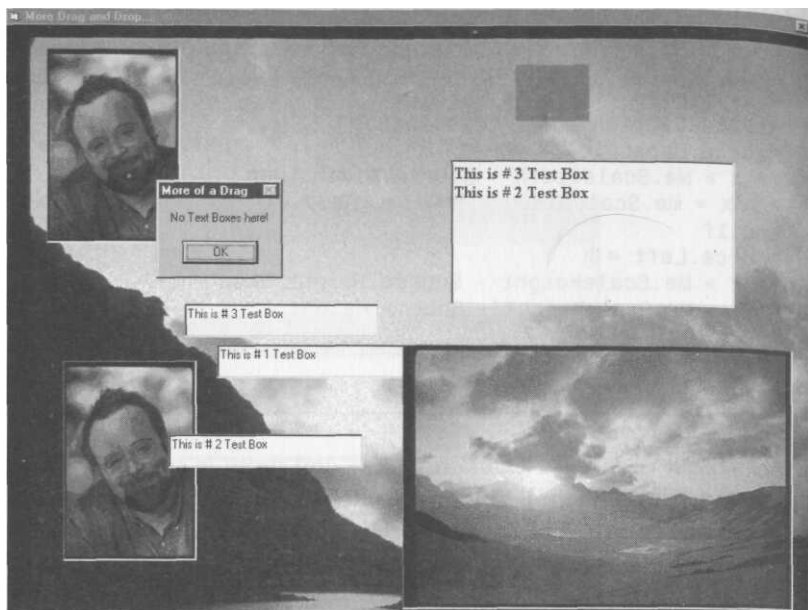
Questa è un'applicazione piuttosto divertente. Come potete vedere, una volta comprese le regole del gioco, la programmazione del drag and drop è decisamente semplice. Passiamo a un altro esempio.

Ancora drag and drop

// prossimo esempio, disponibile sul CD-ROM come Drag2.Vbp, dimostra come aggiungere codice agli eventi DragDrop del codiceper distinguere le diversefonti del trascinamento e agire di conseguenza. Le azioni compiute nel codice sono entro ceni termini simili a quelle che si eseguirebbero con un oggetto OLE.

Controlli Picture

I1 form contiene quattro controlli Picture: due (pctTargetl e pctTarget2) possono essere destinazione per il drag and drop, e due no (pctHarold e Picture1). PctTarget1, pctTarget2, e pctHarold contengono un'immagine (una foto convertita in file .Bmp), e la loro proprietà Autosize è stata impostata a True so in modo che il controllo sia della dimensione esatta dell'immagine che contiene (Figura 21.2).

Figura 21.2
Si può cambiare il contenuto dei controlli Picture e Text utilizzando il drag and drop.

Ecco il codice che determina il tipo di controllo casella immagine *(picture box)* che può essere rilasciato su un altro e copia il contenuto dell'oggetto sorgente nella destinazione quando questo è possibile:

```
'Evento DragDrop. di pctTarget1
pctTarget1.Picture = Source.
PicturepctTarget1.BackColor = vbRed
```

In questo modo si stabilisce che pctTarget1 copi il contenuto della proprietà Picture di qualsiasi controllo Picture. Picturel non ha un'immagine e il suo sfondo è impostato a vbRed: quando si rilascia Picturel su pctTargetl, l'immagine di pctTargetl viene rimossa e il suo sfondo diventa rosso, così il controllo si presenta come un rettangolo rosso (come Picturel). pctTarget2, invece, distingue tra Picturel e gli altri controlli Picture, e non accetta Picturel:

```
'Evento DragDrop di pctTarget2'
If Source.Name = "pctHarold" Then
   pctTarget2.Picture = Source.Picture
Else
   MsgBox "Questo tipo di casella immagine non è accettato!"
End If
```

Sul form sono anche presenti una matrice di caselle di testo e un controllo RichTextBox. Quando si rilascia una casella di testo sulla RichTextBox, il suo contenuto viene accodato a quello del controllo RichTextBox (Figura 21.2). Ecco l'evento DragDrop per il controllo RichTextBox:

Attenzione al titolare dell'evento DragDrop!

Il coodice di questo esempio per l'evento DragOver del form permette a IblDrag di muoversi durante il trascinamento. Questo è un effetto piacevole, ma può portare ad un problema: quando alla fine rilasciate il mouse, lanciando l'evento DragDrop, qesto evento apparterrà all'etichetta, e non al form, perché l'etichetta ha seguito gli sposttamenti del mouse. Il codice dell'esempio risolve questo problema gestendo il rilascio dell'oggetto nell'evento DragDrop dell'etichetta. In questo modo, però, si perde in flessibilità: se fosse possibile scrivere il codice nell'evento DragDrop del form, potremmo gestire le informazioni su sorgente e destinazione in un'unica procedura. Questo è il metodo più comune, ma ovviamente non permette di spostare il controllo nell'evento DragOver. Ecco la procedura DragDrop di un form adattata di conseguenza:

```
Private Sub Form_DragDrop(Source As Control, X As Single, _
    Y As Single)
    Source.Caption = "Sono stata trascinata e rilasciata"
    Source.BackColor = vbBlue
    If X > Me.ScaleWidth - Source.Width Then
        X = Me.ScaleWidth - Source.Width
    End If
    Source.Left = X
    If Y > Me.ScaleHeight - Source.Height Then
        Y = Me.ScaleHeight - Source.Height
    End If
    Source.Top = Y
End Sub
```

Utilizzando un controllo "fantasma" e modificandone le proprietà di visibilità, potremmo anche creare l'impressione di movimento del controllo.

```
Private Sub RichTextBox1_DragDrop(Source As Control, _
    x As Single, y As Single)
    If TypeOf Source Is TextBox Then
        RichTextBox1.Text = RichTextBox1.Text + _
            Source.Text + vbCrLf
    Else
        MsgBox "Accetto solo caselle di testo!"
    End If
End Sub
```

La sintassi If TypeOf... Is controlla se la sorgente è una casella di testo prima di accettare il rilascio. Come si vede nel Listato 21.2, possiamo anche scrivere il codice per gli eventi DragDrop di pctTargetl e pctTarget2 per verificare che su questi controlli non vengano rilasciati altro che controlli Picture.

Listato 21.2 *Sostituzione del contenuto di un controllo con il drag and drop.*

```
Private Sub pctTarget1_DragDrop(Source As Control, x As Single, _
   y As Single)
   If TypeOf Source Is PictureBox Then
      pctTarget1.Picture = Source.Picture
      pctTarget1.BackColor  =  vbRed
   Elself TypeOf Source Is TextBox Then
      MsgBox "Qui non vanno caselle di testo!"
   End If
End Sub

Private Sub pctTarget2_DragDrop(Source As Control, x As Single, _
   y As Single)
   If TypeOf Source Is PictureBox Then
      If Source.Name = "pctHarold" Then
         pctTarget2.Picture = Source.Picture
      Else
         MsgBox "Questo tipo di casella immagine non è accettato!"
      End If
   Elself TypeOf Source Is TextBox Then
      MsgBox "Qui non vanno caselle di testo!"
   End If
End Sub
```

Uso del controllo OLE

Abbiamo già visto come creare automaticamente un contenitore OLE inserendo su un forni il controllo OLE. In effetti, il controllo OLE può essere visto come un controllo personalizzato che mette a disposizione contenitori per oggetti OLE. In altre parole, il controllo contenitore OLE permette di aggiungere ai forni delle applicazioni Visual Basic la possibilità di inserire oggetti. Per ottenere questo risultato si possono seguire molte strade.
Il controllo OLE permette di creare all'interno dell'applicazione uno spazio in cui inserire un oggetto, che può essere a sua volta creato in fase di progettazione utilizzando le finestre di dialogo OLE *Insert Object* e *Paste Special* (presentate nel seguito di questo capitolo) o durante l'esecuzione, impostando opportunamente le proprietà. L'oggetto può essere incorporato o collegato. Utilizzando un controllo Data, è anche possibile collegare il controllo contenitore OLE ad un database. Le Tabelle 21.1 e 21.2 elencano le proprietà e i metodi principali del controllo OLE.

Per avere un elenco completo dei membri del controllo OLE, selezionate la classe OLE nell'Object Browser.

Tahella 21.1 *I metodi principali del controllo OLE Container.*

Metodo	Commento
Close	Chiude un oggetto incorporato e termina la connessione con l' applicazione che l'ha generato.
Copy	Copia un oggetto contenuto in un controllo OLE negli Appunti. Vengono copiate tutte le informazioni incorporate e collegate.
CreateEmbed	Crea un oggetto incorporato basato su un file o una classe. Vedi "Incorporamento o collegamento?" nel seguito di questo capitolo.
CreateLink	Crea un oggetto collegato in base ai contenuti di un file. I parametri di questo metodo sono equivalenti (e hanno la precedenza) alle proprietà SourceDoc e SourceItem.
Delete	Rimuove un oggetto OLE dalla memoria.
DoVerb	Apre un oggetto OLE per una operazione. Vedi la presentazione del metodo DoVerb nel Capitolo 20,.
FetchVerbs	Aggiorna l'elenco delle azioni supportate da un oggetto.
InsertObjDlg	Permette di visualizzare la finestra di dialogo *Insert Object* in fase di esecuzione, in modo che l'utente possa selezionare un oggetto (o un tipo) da inserire nel contenitore OLE.
Paste	Copia i dati dagli Appunti ad un controllo OLE.
PasteSpecialDlg	Visualizza la finestra di dialogo *Paste Special* in fase di esecuzione, in modo che l'utente possa selezionare opzioni come il collegamento o l'incorporamento dell'oggetto contenuto negli Appunti.
ReadFromFile	Carica un oggetto Loads da un file creato dal metodo Save - ToFile.
SaveToFile	Salva un oggetto OLE in un file binario. Se l'oggetto è collegato, vengono salvate solo le informazioni sul collegamento e un'immagine dei dati, mentre i dati dell'oggetto vengono conservati dall'applicazione che lo ha creato. Se l'oggetto è incorporato, i suoi dati vengono conservati dal controllo OLE container e possono essere salvati dall'applicazione Visual Basic.
Update	Aggiorna l'oggetto in un controllo OLE in funzione della sua applicazione di origine.

Se si crea un oggetto incorporato in fase di progettazione, le proprietà SourceDoc e Class (vedi Tabella 21.2) hanno la stessa funzione dei parametri del metodo CreateEmbed. Per conoscere i possibili valori dell'argomento che specifica la classe sul vostro sistema potete selezionare la proprietà Class nella finestra *Properties:* il pulsante a fianco della proprietà Class mostra un elenco dei nomi delle classi disponibili.

Per usare il metodo Paste, impostate la proprietà OLETypeAllowed, quindi controllate il valore della proprietà PasteOK: non è possibile incollare se questo non è **True**.

Se il metodo Paste viene completato, la proprietà OLEType viene impostata a vbOLELinked (= 0) o vbOLEEmbedded (= 1), a seconda che l'oggetto sia collegato o incorporato. Se il metodo fallisce, la proprietà OLEType viene impostata a vbOLENone (= 3).

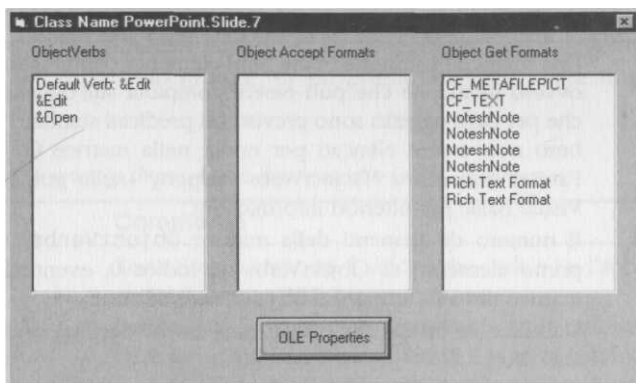
Tabella 21.2 *Leprincipaliproprietà del controllo OLE Container.*

Proprietà	Commento
AppIsRunning	Restituisce o imposta il valore che indica se l'applicazione che ha creato l'oggetto OLE è in esecuzione.
AutoActivate	Il valore della proprietà AutoActivate stabilisce se un oggetto OLE viene attivato a mano (vbOLEActivateManual (0)), con un doppio clic (vbOLEActivateDoubleclick (2)), quando riceve il focus (vbOLEActivateGetFocus (1)), o automaticamente (vbOLEActivateAuto (3)), cioè secondo il metodo predefinito di attivazione dell'oggetto.
AutoVerbMenu	Se AutoVerbMenu vale True, il valore predefinito, facendo clic col pulsante di destra sull'oggetto in fase di esecuzione viene visualizzato un elenco dei predicati applicabili all'oggetto.
Class	Restituisce o imposta il nome della classe di un oggetto OLE (il program ID).
Data	Invia dei dati all'applicazione che ha creato un oggetto.
DataText	Invia o recupera semplice testo da un oggetto OLE.
FileNumber	Il numero del file da utilizzare quando si legge o salva un file.
Format	Restituisce o imposta il formato dei dati inviati o ricevuti dall'applicazione che ha creato l'oggetto.
lpOleObject	Restituisce l'indirizzo dell'oggetto, utilizzato per le chiamate API.
MiscFlags	Imposta o restituisce il valore di un flag che permette di obbligare il controllo OLE a conservare l'oggetto in memoria mentre è caricato e/o modificare la modalità di attivazione sul posto rispetto al default per gli oggetti che la prevedono.
Object	Permette di specificare un oggetto di cui si vogliono utilizzare proprietà e metodi in un task di automazione OLE.
ObjectAcceptFormats	Una matrice di stringhe in cui ogni elemento descrive un formato accettabile per la proprietà Format durante lo scambio di dati con un oggetto mediante le proprietà Data e DataText.
ObjectAcceptFormats-Count	Il numero di elementi della matrice ObjectAcceptFormats. Visto che il primo elemento di ObjectAcceptFormats ha indice O, eventuali cicli sulla matrice devono fermarsi a ObjectAcceptFormatsCount -1.
ObjectGetFormats	Una matrice di stringhe in cui ogni elemento descrive un formato di dati generabile dall'oggetto.
ObjectGetFormats-Count	Il numero di elementi della matrice ObjectGetFormats. Visto che il primo elemento di ObjectGetFormats ha indice O, eventuali cicli sull'array devono fermarsi a ObjectGetFormatsCount - 1.
ObjectVerbFlags	Restituisce lo stato del menu per un predicato della matrice ObjectVerbs.

Proprietà	Commento
Objec tVerbs	Una matrice di stringhe in cui ogni elemento contiene un predicato, ovvero un'azione che può essere compiuta sull'oggetto. Ricordate che per ogni oggetto sono previsti sei predicati standard che potrebbero non essere elencati per nome nella matrice ObjectVerbs. Fate riferimento a "ObjectVerbs Property" nella guida in linea di Visual Basic per ulteriori informazioni.
ObjectVerbsCount	Il numero di elementi della matrice ObjectVerbs. Visto che il primo elemento di ObjectVerbs ha indice O, eventuali cicli sulla matrice devono fermarsi a Obj ectVerbsCount - 1.
OLEDropAllowed	Stabilisce se un controllo OLE può essere destinazione di un drag and drop.
OLEType	Indica se un controllo OLE contiene un oggetto collegato, incorporato o nessun oggetto.
OLETypeAllowed	Imposta la possibilità per un controllo OLE di contenere un oggetto collegato, incorporato o nessun oggetto.
PasteOK	Indica se il contenuto degli Appunti può essere incollato in un controllo OLE.
SizeMode	Controlla la visualizzazione di un oggetto all'interno del contenitore.
SourceDoc	Restituisce o imposta il nome del file da utilizzare per la creazione di un oggetto.
SourceItem	Restituisce o imposta i dati all'interno del file da collegare quando si crea un oggetto collegato.
UpdateOptions	Restituisce o imposta un valore che specifica come avviene l'aggiornamento di un oggetto quando cambiano i dati collegati. I valori possibili sono: vbOLEAutomatic (valore predefinito, 0). L'oggetto viene aggiornato ogni volta che cambiano i dati collegati. VbOLEFrozen (1). L'oggetto viene aggiornato ogni volta che l'utente salva i dati collegati dall'applicazione che l'ha creato. VbOLEManual (2). L'oggetto viene aggiornato solo dal metoo Update.

Vediamo alcune di queste proprietà all'opera. Questo esempio, disponibile sul CD-ROM come Verbs.Vbp, visualizza il contenuto delle matrici ObjectVerbs, ObjectAcceptFormats, *e* Obj ectGetFormat *dell'oggetto OLE selezionato dall'utente (Figura 21.3)-*

Il nome della classe cui appartiene l'oggetto viene visualizzato nella didascalia del rorm dell'esempio. Notate che il predicato predefinito compare due volte nella matrice ObjectVerb, la prima come elemento 0. Il Listato 21.3 mostra le azioni e i formati dell'oggetto OLE.

Figura 21.3 *Le informazioni su un oggetto OLE (nella Figura, l'oggetto è una diapositiva PowerPoint) possono essere facilmente recuperate utilizzando i metodi del controllo OLE container.*

Llstato 21.3 *Elenco dei predicati (Verbs) e dei formati.*

```
Private Sub cmdInsert_Click()
   Dim I As Integer
    ' Visualizza la finestra di dialogo Insert Object.
   Ole1.InsertObjDlg
   Form1.Caption = "Nome della Classe " + Ole1.Class
   Ole1 .FetchVerbs   ' Carica i verbi.

   lstVerb.Clear
   lstAccept.Clear
   lstGet.Clear

    ' Riempie la casella di riepilogo dei verbi.
   lstVerb.AddItem "Verbo di defaul: " + Ole1.ObjectVerbs(O)
   Por I = 1 To Ole1.ObjectVerbsCount - 1
      lstVerb.AddItem Ole1.ObjectVerbs(I)
   Next I

    'Riempie la casella di riepilogo dei formati accettati.
   Por I = 0 To Ole1 .ObjectAcceptFormatsCount - 1
      lstAccept.AddItemOle1.ObjectAcceptFormats(I)
   Next I

     ' Riempie la casella di riepilogo dei formati presi.
   Por I = 0 To Ole1 .ObjectGetFormatsCount - 1
      lstGet.AddItem Ole1.ObjectGetFormats(I)
   Next I
End Sub
```

*Il prossimo esempio, disponibile sul **CD-ROM** come Link1.Vbp, esegue delle operazioni su un documento Wordper Windows collegato ad un controllo OLE container.*

Il documento da collegare è salvato come Source.Doc sul CD-ROM. Per poter seguire questo esempio dovrete copiare il file sul vostro hard disk, e ovviamente avere una versione di Word per Windows installata nel sistema. Per collegare il documento, aggiungete al vostro forni un controllo contenitore OLE: si aprirà la finestra di dialogo Insert Object *(vedi Figura 21.4). Selezionate il pulsante* Create From File: *compariranno la casella Link ed il pulsante Browse, che permette di localizzare il file da collegare. Una volta selezionato il file, attivate la casella* Link.

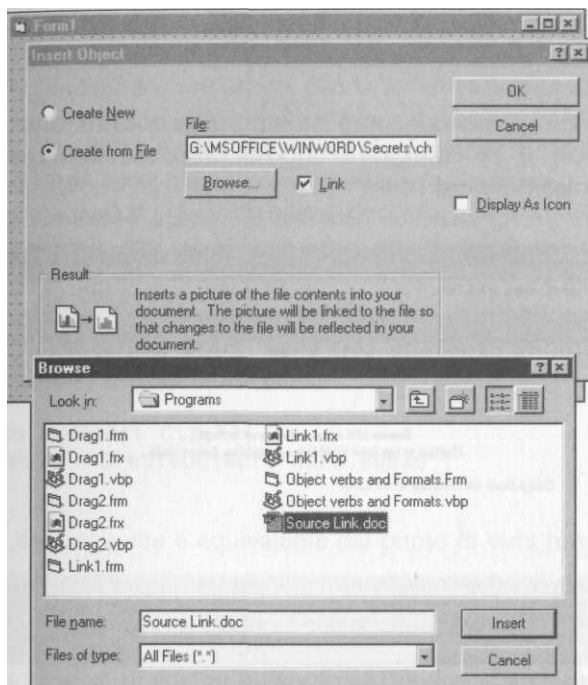
Figura 21.4 *Per collegare un oggetto a un controllo OLE container di Visual Basic si può usare la finestra di dialogo Inserì Object.*

Verificate che la proprietà del controllo OLE sia impostata a 1 -`Stretch` in modo che l'intero documento Word sia rappresentato nell'area del controllo OLE (Figura 21.5). L'impostazione predefinita mostrerebbe solo una piccola parte del documento. Il documento Source contiene un'intestazione e un segnalibro che permette all'applicazione Visual Basic di posizionarsi nel punto in cui verrà inserito il contenuto di una casella di testo. Il segnalibro si chiama "animal" perché l'intestazione è diretta agli animali (Figura 21.5).

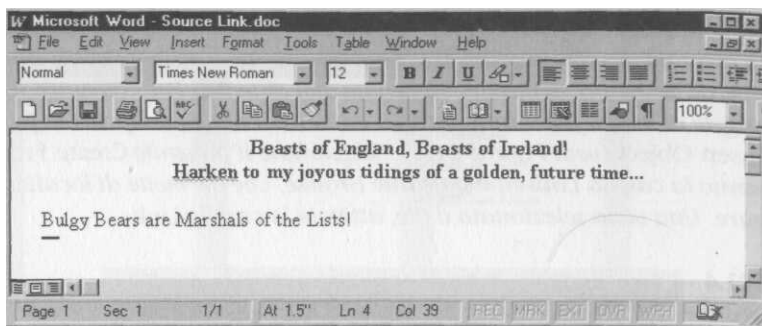
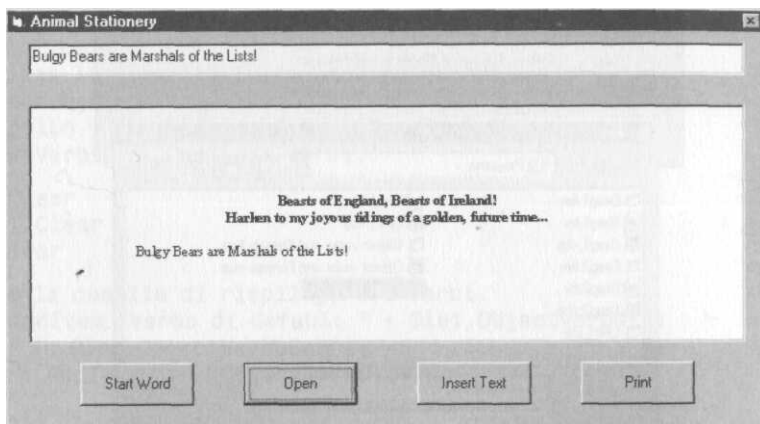
Figura 21.5
Questo è il documento originale Word per Windows collegato.

Il programma d'esempio presenta quattro pulsanti (Figura 21.6). *Start Word* avvia Word per Windows con il file collegato e imposta una variabile oggetto che contiene un'istanza dell'oggetto Word Basic.

Figura 21.6
Usando il controllo OLE è facile collegare un 'applicazione Visual Basic di un documento Word per Windows.

Il pulsante *Open* apre il documento Word collegato nel suo ambiente nativo per la modifica. Questo non sarebbe necessario se ci si limita ad applicare i metodi dell'automazione OLE all'oggetto WordBasic. Per esempio, si potrebbe utilizzare un'istruzione come

```
objWord.FileOpen OLE1.SourceDoc 'objWord è un oggetto WordBasic
```

per aprire il file collegato per eseguire ulteriori azioni di automazione OLE senza visualizzare l'ambiente Word. Il pulsante *Insert Text* copia il testo del controllo Text dell'applicazione VB nella posizione indicata dal segnalibro "animai" del documento collegato. Il pulsante *Prìnt* stampa il documento Word per Windows.
Se si usa il metodo DoVerb, le azioni che si possono svolgere su un oggetto all'interno del controllo OLE sono limitate ai "verbi" che fornisce. Word per Windows non fornisce un verbo per inserire del testo né uno per stampare i documenti. (Se lanciate l'applicazione Object Verbs and Formats vista prima su Word, troverete due verbi: Edit e Open). Sicuramente non c'è un verbo che lanci una macro: per farlo è necessario referenziare l'oggetto WordBasic.

La fonte migliore di informazioni sulla programmazione degli oggetti OLE di Word è l'argomento "Using OLE Automation with Word" nella guida in linea di Word, che è un sottoargomento di "More Word Basic Information ", a sua volta contenuto in "Word Basic Reference".

A partire dalla versione 8 di Word, contenuta in Office 97, per i suoi oggetti OLE è previsto un nuovo modello, decisamente migliorato. Per esempio, l'oggetto solitamente chiamato alla radice della gerarchla è Word.Application, anziché che Word.Basic, la gerarchla di Office 97 viene ripresa in Office 2000. Per approfondimenti sulla programmazione degli oggetti esposti da Office 97, si veda il Capitolo 22.

Probabilmente vi farà piacere sapere che la struttura degli oggetti di Word 7 è stata mantenuta per assicurare la compatibilita con il passato. In altre parole, codice VB che faccia riferimento a oggetti di Word. Basic funzionerà correttamente anche sotto Office 97 e Office 2000. Tanto per cominciare, dichiariamo a livello di forni una variabile che conterrà l'oggetto WordBasic:

```
Option Explicit
Dim objWord As Object
```

Il codice della procedura Start Word crea un'istanza di objWord in modo che si possano chiamare i metodi di WordBasic:

```
Private Sub cmdStart_Click()
   Set objWord = CreateObject("Word.Basic")
End Sub
```

L'istruzione Set seguente è equivalente dal punto di vista funzionale a quella della procedura:

```
Set objWord = OLE1 .object.Application.WordBasic
```

Per aprire l'ambiente di editing di Word per Windows, si chiama il metodo DoVerbs del controllo OLE passando come argomento O (Edit):

```
Private Sub cmdOpen_Click()
   OLE1.DoVerb (0)
   Me.SetFocus
End Sub
```

La procedura restituisce il focus all'applicazione VB dopo che Word è stato avviato. Saltare al segnalibro e inserire il testo contenuto in Textl è banale:

```
Private Sub cmdInsert_Click()
   objWord.EditGoto "Animai"
   objWord.Insert Text1.Text
End Sub
```

Per chiamare il comando di stampa di WordBasic per il documento collegato basta:

```
Private Sub cmdPrint_Click()
   objWord.FilePrint
End Sub
```

Alla chiusura dell'applicazione VB, è importante che venga liberata la variabile oggetto e che venga chiuso Word per Windows:

Uso di DDE al posto di OLE

Il DDE (Dynamic Data Exchange) sembra ormai una tecnologia antica. In realtà fino a non molto tempo fa era il modo migliore per far comunicare le applicazioni Windows. Scoprirete che a volte può ancora tornare utile, in particolare se avrete a che fare con applicazioni datate. Questo è l'equivalente DDE della procedura OLE:

```
Text1.LinkMode = vbLinkNone
Text1.LinkTopic = "WinWord]" + OLE1.SourceDoc
Text1.LinkItem = "Animal"
Text1.LinkMode = vbLinkManual
Text1.LinkPoke
Text1.LinkMode = vbLinkNone
```

```
Private Sub Form_Unload(Cancel As Integer)
   OLE1.Close 'Chiude Word
   Set objWord = Nothing
End Sub
```

Ci sono due avvertimenti del metodo OLE Close: il primo è che se Word non è stato avviato dal controllo OLE (per esempio, perché era già in esecuzione prima della chiamata al metodo DoVerbs) il metodo dose del controllo OLE non potrà chiuderlo.

// secondo è che eventuali modifiche al documento collegato non saranno salvate automaticamente: dovrete occuparvene voi, chiamando i metodi FileClose *o* FileSave *di WordBasic prima che venga chiamato il metodo* dose *del controllo OLE. Ecco il codice completo dell'applicazione di esempio:*

```
Option Explicit
Dim objWord As Object

Private Sub cmdStart_Click( )
   Set objWord = CreateObject("Word.Basic")
End Sub

Private Sub cmdOpen_Click()
   OLE1.DoVerb (0)
   Me.SetFocus
End Sub

Private Sub cmdInsert_Click()
   objWord.EditGoto "Animal"
   objWord.Insert Text1.Text
End Sub

Private Sub cmdPrint_Click()
   objWord.FilePrint
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
    OLE1.Close   'Close Word
    Set objWord = Nothing
End Sub
```

Incorporamento o collegamento?

Gli oggetti incorporati sono interamente contenuti nel form e all'interno dell'applicazione VB. Questo comporta i seguenti vantaggi:

I dati dell'oggetto sono completamente sotto il controllo dell'applicazione e non possono essere rinominati, spostati o cancellati da altre applicazioni.

L'oggetto incorporato viene installato automaticamente insieme all'eseguibile dell'applicazione.

Alcuni svantaggi dell'incorporamento sono:

- La dimensione dell'eseguibile aumenta come effetto dell'inclusione dei dati dell'oggetto.
- L'applicazione gestisce una propria copia dell'oggetto incorporato, quindi eventuali modifiche vengono apportate solo a questa e non ad altre copie. Gli oggetti incorporati non possono essere gestiti centralmente in modo che le modifiche ad una copia si propaghino alle altre.
- Una volta che un oggetto incorporato viene inserito in un controllo *OLE,* l'unico modo per compiere azioni su di esso è utilizzare i verbi (le azioni) OLE previsti. Per esempio, è possibile realizzare un collegamento tra il contenuto di una casella di testo e l'oggetto incorporato solo se questo prevede un verbo che lo permette. Questo è il motivo per cui, nell'esempio precedente, il documento Word per Windows è stato collegato e non incorporato.
- Quando viene attivata, l'applicazione che ha creato l'oggetto incorporato può lavorare esclusivamente su di esso.

Dal punto di vista degli oggetti collegati, i vantaggi rispetto all'incorporamento sono:

- Le modifiche apportate ai dati collegati si propagano a tutte le copie collegate, rendendo possibile la gestione centralizzata degli oggetti collegati.
- La dimensione dell'eseguibile non aumenta come prima, perché vengono incluse solo le informazioni sul collegamento e non l'intero insieme di dati.

L'insieme delle azioni ammesse dall'oggetto collegato può essere molto più ampio perché non si è più limitati ai verbi OLE previsti dall'oggetto.

Gli svantaggi del collegamento rispetto all'incorporamento sono:

L'applicazione può fallire e presentare un oscuro messaggio di errore ("Unable to activate object") se il file dei dati viene spostato, rinominato o eliminato (Figura 21.7).

Figura 21.7
Tipico scherzo a un 'applicazione collegata: il file è stato spostato!

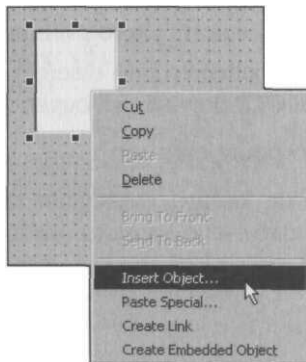
- L'attivazione di un oggetto collegato apre l'applicazione in un proprio spazio di lavoro invece che per la modifica in loco. Questo potrebbe consumare più risorse e dare all'utente un controllo eccessivo sui controlli nativi dell'applicazione.

Uso del menu di scelta rapida del contenitore OLE

Quando si inserisce su un forni un nuovo controllo OLE container, viene aperta la finestra di dialogo *Insert Object,* che permette di creare un oggetto collegato o incorporato (come vedremo tra poco). Se si sceglie *Cancel,* non viene creato alcun oggetto. In fase di progettazione, facendo clic col pulsante di destra sul contenitore OLE si visualizza il menu riprodotto in Figura 21.8: i comandi disponibili nel menu di scelta rapida dipendono dallo stato del contenitore OLE, come dettagliato nella Tabella 21.3.

Tabella 21.3 *Disponibilità dei comandi nel menu di scelta rapida del controllo OLE.*

Comando	Disponibile quando
Insert Object	Sempre
Paste Special	Quando negli Appunti è presente un oggetto valido
Delete Embedded Object	Quando il controllo OLE container contiene un oggetto incorporato
Delete Linked Object	Quando il controllo contenitore OLE contiene un oggetto collegato
Create Link	Quando è impostata la proprietà SourceDoc del controllo
Create Embedded Object	Quando è impostata la proprietà Class (vedi oltre) o SourceDoc del controllo

Figura 21.8
// menu di scelta rapida del contrmicro OLE permette di inserire oggetti e creare o eliminare collegamenti.

Creazione di oggetti in fase di progettazione

Ogni controllo contenitore OLE può contenere un solo oggetto per volta: questo oggetto che sia collegato o incorporato, può essere creato in diversi modi:

Utilizzando le finestre di dialogo *Insert Object* o *Paste Special* (in esecuzione o in fase di progettazione)

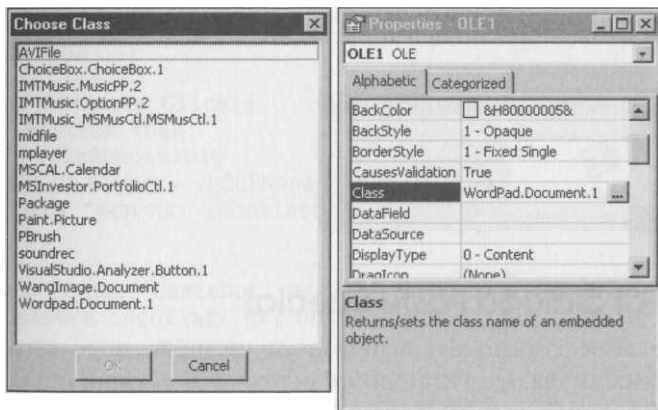
Impostando la proprietà Class nella finestra *Properties,* quindi usando il menu di scelta rapida del controllo OLE (solo in fase di progettazione)

Utilizzando i metodi del controllo OLE in fase di esecuzione

Sostanzialmente, in fase di progettazione abbiamo a disposizione in tutto tre metodi per creare un oggetto: usare la finestra di dialogo *Insert Object,* usare la *Paste Special,* o impostare una classe.

Trovare i nomi delle classi

Per ottenere dal controllo OLE un elenco dei nomi delle classi disponibili si seleziona la proprietà Class del controllo OLE container nella finestra *Properties* e si fa clic sul pulsante *Properties* (vedi Figura 21.9).

Figura 21.9
La proprietà Class del controllo OLE elenca i nomi delle classi disponibili.

La finestra di dialogo Insert Object

La finestra di dialogo *Insert Object* propone due scelte importanti:

L'oggetto deve essere collegato o incorporato? Abbiamo appena visto le differenze tra i due. Notate che un oggetto nuovo non può essere collegato: ciò è possibile solo con oggetti creati da file. (È possibile, comunque, creare un collegamento in un secondo tempo.) La Figura 21.4 riproduce la

finestra di dialogo *Insert Object* nel caso di un oggetto collegato; la Figura 21.10 mostra la finestra nel caso di un oggetto incorporato.

- **L'oggetto è nuovo o deve essere creato da un file?** Oltre al fatto che non è possibile collegare un file nuovo, è importante tenere presente un altro aspetto: se volete che l'oggetto contenga dei dati di default (per esempio il titolo e il segnalibro dell'esempio di prima) dovreste crearlo basandovi su un file esistente.

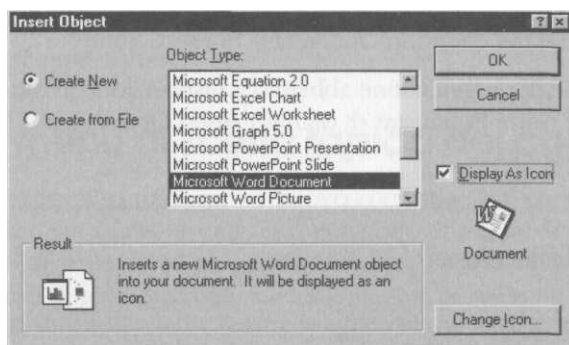
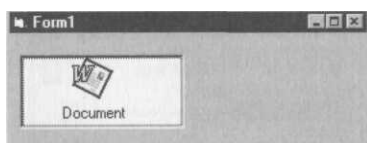
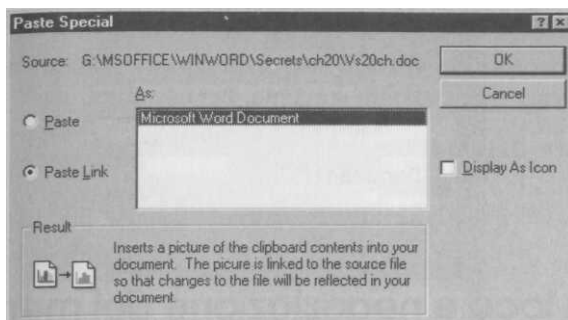
Figura 21.10
La finestra di dialogo Insert Object permette di creare un nuovo documento incorporato.

Inoltre, è possibile scegliere (come in Figura 21.10) di visualizzare l'oggetto come icona (vedi Figura 21.11): in questo caso, anche se l'attivazione in loco continua ad aprire l'applicazione dell'oggetto, questo si presenta come icona piuttosto che come immagine dello spazio di lavoro dell'applicazione.

Figura 21.11
Gli oggetti che compaiono come iconepossono essere attivati in loco.

La finestra di dialogo Paste Special

Una volta che avete copiato del materiale da un'applicazione server OLE agli Appunti, la finestra di dialogo *Paste Special* permette di incorporarlo o collegarlo a un controllo OLE container (vedi Figura 21.12). La finestra di dialogo *Paste Special* torna particolarmente utile quando volete incorporare o collegare solo una parte di un file, come un paragrafo di un documento Word per Windows. In fase di progettazione, la finestra di dialogo *Paste Special* è accessibile dal menu di scelta rapida del controllo OLE.

Impostazione della classe dell'oggetto

L'ultimo metodo per riempire un contenitore OLE in fase di progettazione è impostarne la proprietà Class e quindi selezionare *Create Embedded Object* dal menu di scelta rapida.

Figura21.12

Creazione di oggetti in fase di esecuzione

Per visualizzare la finestra di dialogo *Insert Object* per permettere all'utente di selezionare le opzioni, si usa il metodo InsertObjDlg del controllo OLE:

```
Private Sub cmdInsert_Click()
   OLE1.InsertObjDlg
   If OLE1.OLEType = vbOLENone Then
      MsgBox "Non hai creato un oggetto!"
   End If
End Sub
```

La finestra di dialogo *Paste Special* funziona allo stesso modo, chiamando il metodo PasteSpecialDlg:

```
Private Sub cmdPaste_Click()
   If OLE1.PasteOK Then
      OLE1.PasteSpecialDlg
      If OLE1.OLEType = vbOLENone Then
         MsgBox "Non hai incollato un oggetto!"
      End If
   Else
      MsgBox "I dati contenuti negli Appunti non possono" + _
         " essere incollati nel controllo OLE"
   End If
End Sub
```

Uso dei metodi del controllo OLE

Per creare un oggetto collegato in fase di esecuzione si può usare il metodo CreateLink del controllo OLE:

```
OLE.CreateLink"C:\Secrets\ch21\programs\Source.doc"
```

La proprietà .SourceItem permette di specificare quali dati all'interno del file si vogliono collegare. Per creare un oggetto incorporato in fase di esecuzione si usa il metodo .CreateEmbed:

```
OLE1.CreateEmbed  "C:\Secrets\ch21\programs\Source.doc"
```

Per creare un oggetto incorporato vuoto in fase di esecuzione, si usa il metodo CreateEmbed senza specificare il documento sorgente. Per esempio:

```
Private Sub cmdCreate_Click()
   OLE1.CreateEmbed "", "Word.Document"
   OLE1.DoVerb 0
End Sub
```

Attivazione in loco e negoziazione dei menu

Se volete che il menu dell'oggetto compaia sul vostro form dopo l'attivazione in loco (Figura 21.13), dovete aggiungere al form almeno una voce di menu, non necessariamente invisibile.

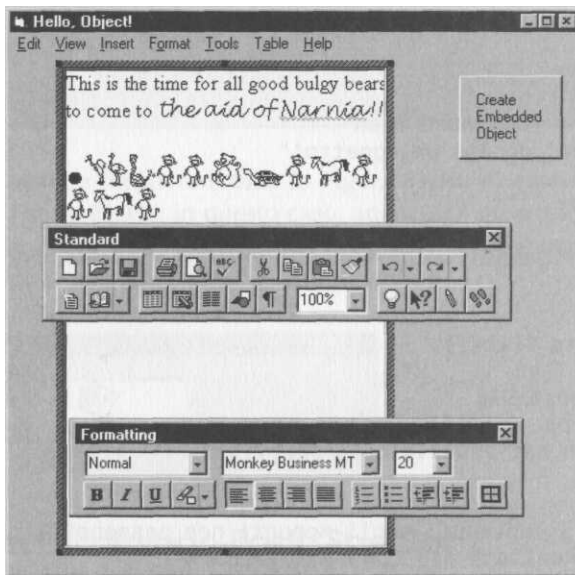
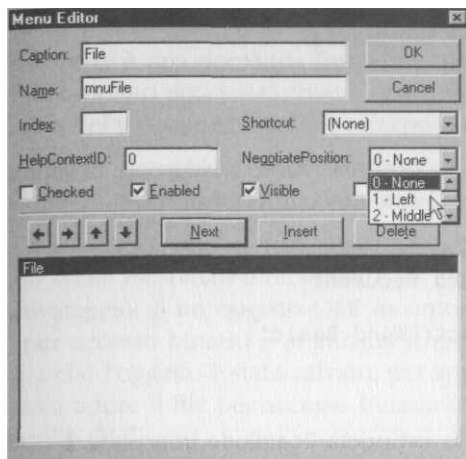
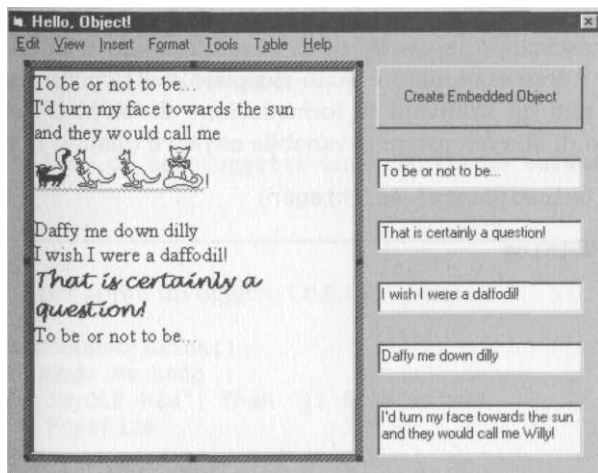
Figura 21.13 *Per fare in modo che il menu di un oggetto attivato sul posto compaia nel form di un controllo OLE, assicuratevi che il form abbia almeno una voce di menu.*

La negoziazione dei menu stabilisce quali menu debbano comparire in caso di richieste di spazio nel menu in competizione. Questo argomento è già stato affrontato in maggior dettaglio nel Capitolo 18. Come già visto nel Capitolo 18, è possibile impostare la proprietà *NegotiatePosition* di una voce di menu nel *Menu Editar* (vedi Figura 21.14).

Figura 21.14 *IL MenuEditar permette di impostare la negoziazione dei menu.*

Se *NegotiatePosition* viene impostata a 0-None, l'elemento originale del menu del form scomparirà quando l'applicazione dell'oggetto OLE verrà attivata. In caso contrario, la proprietà *NegotiatePosition* stabilisce dove compare la voce di menu originale.

Drag and drop su controlli OLE

All'inizio del capitolo abbiamo parlato di programmazione del drag and drop, e in conclusione, per chiudere il cerchio, vedremo una dimostrazione di drag and drop nel contesto di un controllo OLE.

La dimostrazione, disponibile sul CD-ROM come Embed.Vbp, permette all'utente di trascinare e rilasciare caselle di testo su un controllo OLE contenente un documento incorporato Wordper Windows vuoto (vedi Figura 21.15). (La procedura per creare un oggetto Word incorporato vuoto è stata già descritta in questo capitolo parlando dell'uso dei metodi del controllo OLE infase di esecuzione.)

Figura 21.15 *Le tecniche di programmazione per il drag and drop funzionano benissimo anche con OLE?*

Ecco come funziona questo progetto. Innanzi tutto, viene dichiarata una variabile che conterrà l'istanza dell'oggetto WordBasic:

```
Option Explicit
Dim objWord As Object
```

Per creare un documento Word incorporato vuoto e istanziare la variabile objWord l'utente può premere il pulsante *Create Embedded Object:*

```
Private Sub cmdCreate_Click()
   OLE1.CreateEmbed "", "Word.Document"
   OLE1.DoVerb 0
   Set objWord = CreateObject("Word.Basic")
   Me.SetFocus
End Sub
```

Tutte le caselle di testo hanno la proprietà DragMode impostata a 1 -Automatic; la proprietà DragIcon è impostata con una delle icone presenti nella libreria di Visual Basic (un fulmine).
Quando si rilascia una casella di testo sul controllo OLE, il codice dell'evento DragDrop verifica se il controllo contiene un oggetto incorporato: in caso contrario, ne viene creato uno e viene istanziata la variabile objWord. In ogni caso, la proprietà text del controllo rilasciato viene accodata all'oggetto Word per Windows incorporato:

```
Private Sub OLE1_DragDrop(Source As Control, X As Single, _
   Y As Single)
   On Error Resume Next
   If Not OLE1.OLEType = vbOLEEmbedded Then
      OLE1.CreateEmbed "", "Word.Document"
      OLE1.DoVerb 0
      Set objWord = CreateObject("Word.Basic")
      Me.SetFocus
   End If
   objWord.Insert Source.Text + vbCrLf
End Sub
```

Se giocate un po' trascinando le caselle di testo sul controllo OLE, vi accorgerete che, una volta che il loro contenuto è stato ricopiato nell'oggetto incorporato, è possibile utilizzare tutti gli strumenti di formattazione di Word (Figura 21.15). È importante ricordarsi di dereferenziare la variabile objWord quando non serve più:

```
Private Sub Form_Unload(Cancel As Integer)
   OLE1 .close
   Set objWord = Nothing
End Sub
```

Il metodo SaveToFile

Essere o non essere: è una domanda frequente, parlando di oggetti. Visto che un oggetto OLE incorporato non sopravvive al suo contenitore, cosa fate se volete salvare il contenuto del vostro oggetto OLE incorporato?

Gli oggetti collegati dovrebbero essere salvati utilizzando gli appositi comandi dell'applicazione di attivazione.

Il controllo OLE fornisce due metodi (SaveToFile e ReadFromFile) che permettono di salvare come file binari e recuperare gli oggetti OLE incorporati. La procedura per il salvataggio di un oggetto OLE incorporato è decisamente semplice: si apre un file per accesso binario e si chiama il metodo SaveToFile del controllo OLE. Una volta che l'oggetto è stato salvato, per aprirlo e visualizzarlo in un contenitore OLE basta aprire il file per accesso binario e chiamare il metodo ReadFromFile del controllo OLE.
Ho aggiunto due voci al menu *File* del Form1 del progetto Embed.Vbp: mnuSave e mnuOpen. Notate che *NegotiatePosition* per il menu *File* è stata impostata a 1-Left in modo che i menu *Save e Open* compaiano dopo l'attivazione sul posto e la negoziazione dei menu. Ecco il codice principale per salvare un oggetto OLE incorporato:

```
Dim FileNumber As Long
FileNumber = FreeFile
Open "MyOLE.Hld" For Binary As #FileNumber
OLE1.SaveToFile FileNumber
dose #FileNumber
```

Sarebbe bene aggiungere qualche controllo, come si vede nel Listato 21.4: abbiamo veramente qualcosa da salvare?

Listato 21.4 *Uso del metodo SaveToFile.*

```
Private Sub mnuSave_Click()
   Dim FileNumber As Long
   If OLE1.OLEType = vbOLEEmbedded Then
      FileNumber = FreeFile
      Open "MyOLE.Hld" For Binary As #FileNumber
      OLE1.SaveToFile FileNumber
      Close #FileNumber
   Else
      MsgBox "Non ci sono oggetti incorporati da salvare!"
   End If
End Sub
```

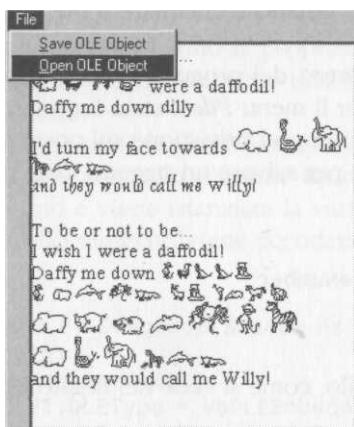
Ecco il codice per aprire un oggetto OLE incorporato.

```
Private Sub mnuOpen_Click()
   Dim FileNumber As Long
   If Exists("MyOLE.Hld") Then 'il file esiste
FileNumber = FreeFile
```

```
        Open "MyOLE.Hld" Por Binary As #FileNumber
        OLE1.ReadFromFile  FileNumber
        Close  #FileNumber
    Else
        MsgBox "Non c'è niente da aprire!"
    End If
End Sub
```

Le procedure Open e Save funzionano bene, ma potreste migliorarle utilizzando una finestra di dialogo comune per impostare il nome del file da aprire o salvare.

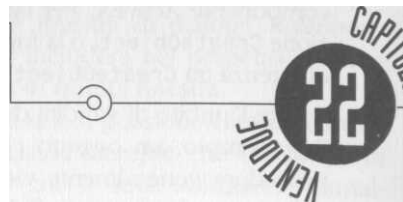
Sul CD-ROM troverete il MyOLE.Hld, che è stato creato con questa dimostrazione utilizzando il controllo OLE con un oggetto incorporato. Per aprirlo potete usare il programma dimostrativo (vedi Figura 21.16).

Figura 21.16 *Salvare il contenuto di oggetti OLE incorporati in file binarì è facile.*

Riepilogo

I temi di questo capitolo erano due: la programmazione del drag and drop, e il funzionamento del controllo contenitore OLE. Alcuni progetti dimostrativi hanno evidenziato le tecniche per la gestione del drag and drop.

- Abbiamo visto come abilitare e disabilitare il trascinamento in fase di esecuzione.
- Abbiamo visto come visualizzare lo spostamento degli oggetti.
- Abbiamo visto come usare il drag and drop per sostituire il contenuto di un controllo.
- Abbiamo esaminato il rilascio di un controllo su un oggetto OLE incorporato.
- Abbiamo esaminato a fondo il controllo contenitore OLE.
- Abbiamo esaminato metodi e proprietà del controllo OLE.
- Abbiamo introdotto il menu di scelta rapida del controllo OLE.

Abbiamo introdotto verbi (azioni) e formati degli oggetti.
Abbiamo visto che cosa sia l'attivazione in loco.
Abbiamo confrontato collegamento ed incorporamento.
Abbiamo visto come creare oggetti in fase di progettazione.
Abbiamo visto come creare oggetti in fase di esecuzione.
Abbiamo introdotto il tema della negoziazione dei menu OLE.
Abbiamo visto come salvare il contenuto di un oggetto OLE incorporato.

CONTROLLO DI OGGETTI DI APPLICAZIONI ESTERNE

- Lavorare con componenti ActiveX
- Scorrere le gerarchie degli oggetti
- Usare Excel per calcolare gli interessi sui prestiti
- Inserire un controllo personalizzato Excel
- Usare Excel come correttore ortografico per Visual Basic
- Creare e modificare documenti Word
- Modificare un database Access

Nel Capitolo 21, abbiamo visto come usare il controllo contenitore OLE per collegare o incorporare in un progetto Visual Basic oggetti di altre applicazioni. In questo capitolo continueremo il discorso, facendo un piccolo passo avanti: parleremo infatti di come usare l'automazione OLE e l'interfaccia ActiveX per manipolare da VB gli oggetti esposti da un'altra applicazione senza utilizzare il controllo OLE.
Nella vita reale, i peggiori grattacapi derivanti dall'uso di OLE da Visual Basic per controllare componenti ActiveX esterni riguardano la comprensione di quali siano gli oggetti esposti dall'applicazione e la ricerca delle informazioni su come utilizzarne metodi e proprietà. Le nuove applicazioni, come nel caso di Office 97, diventano sempre più rigorosamente orientate agli oggetti rispetto alle vecchie versioni, il che semplifica le cose. L'Object Browser di Visual Basic permette di esplorare la struttura delle applicazioni ActiveX, una volta che si è aggiunto al progetto un riferimento alla relativa libreria. Può anche servire avere a disposizione la documentazione di sviluppo, un Software Development Kit (SDK) o un Resource Kit, per la particolare applicazione ActiveX.
Gli esempi di questo capitolo dimostrano come usare Access, Excel, e Word come componenti ActiveX esterni da Visual Basic. (Un tempo si sarebbe detto che le applicazioni venivano usate come server OLE.)

I programmi dimostrativi in questo capitolo sono stati preparati e testati con Office 97 Professional, che comprende la versione 8 di Access, Excel e Word. Alcune delle tecniche utilizzate potrebbero funzionare anche con versioni OLE precedenti (o successive) delle stesse applicazioni, ma potrebbero essere necessarie piccole modifiche alla sintassi o ai nomi degli oggetti.

Lavorare con componenti ActiveX

Come vedremo negli esempi di questo capitolo, per controllare con successo un componente ActiveX esterno è necessario:

- Capire come creare ed avviare un'istanza dell'oggetto desiderato, esposto dal componente ActiveX. Per fare questo si usano la parola chiave New, la funzione CreateObject, o la funzione GetObject. (Il riquadro in basso spiega la differenza tra CreateOb] ect e GetObject).
- Capire l'ambito di validità dell'oggetto all'interno della vostra applicazione. Per esempio, un oggetto che viene istanziato (creato) all'interno di una procedura generalmente viene distrutto quando cessa l'ambito di validità della procedura stessa. È buona norma rilasciare una variabile oggetto (impostandola a Nothing) se all'interno del suo ambito di validità non serve più.
- Conoscere la gerarchia dell'oggetto. Quali sono i membri esposti dall'oggetto disponibili ai client attraverso l'automazione ActiveX? Gli oggetti all'interno di altri oggetti (detti "sotto-oggetti" o "oggetti subordinati") devono normalmente essere inizializzati da un metodo dell'oggetto che li precede nella gerarchia.
- Conoscere i metodi che l'oggetto ActiveX e i suoi sotto-oggetti espongono, e la relativa sintassi. Il modo migliore per recuperare queste informazioni è utilizzare PObject Browser, o ottenere la documentazione dell'applicazione server ActiveX.

GetObject o CreateObject?

La funzione GetObject dovrebbe essere utilizzata se è già stata creata un'istanza dell'oggetto, o se si vuole creare l'oggetto con un file precaricato (per esempio, un oggetto Word con un documento già aperto). Se non esiste ancora un'istanza dell'oggetto, e non si vuole avviare l'oggetto con un file già aperto, si usa la funzione CreateObject.
Notate che se un oggetto si è registrato come oggetto a singola istanza, verrà creata una sola istanza dell'oggetto indipendentemente da quante volte verrà chiamato Cre - ateObject. Nel caso di oggetti a singola istanza, GetObject restituisce sempre la stessa istanza quando viene chiamato con una stringa nulla (""), e un errore se si omette il percorso dell'oggetto. Non è possibile utilizzare GetObj ect per ottenere un riferimento a una classe creata con Visual Basic.

Referenziare un oggetto per cui è disponibile una libreria di oggetti

TJna libreria di oggetti fornisce una descrizione di un'applicazione componente ActiveX elencando tutti gli oggetti definiti ed i relativi membri. (I membri di un oggetto sono la sua interfaccia, cioè proprietà, eventi e metodi). Tutti gli oggetti per cui è disponibile una libreria sono elencati nella finestra di dialogo *References* (accessibile dal menu *Project* di Visual Basic). Per includere nel progetto corrente una libreria di oggetti, verificate che sia selezionata in questa finestra.

I metodi e le proprietà degli oggetti di una libreria che è possibile creare possono essere istanziati utilizzando l'identificatore della classe, sempre che la libreria di oggetti sia stata inclusa nel progetto Visual Basic che li deve utilizzare. Normalmente si usa New per creare un'istanza di oggetti la cui libreria di oggetti è inclusa nel progetto, mentre si usano CreateObject e Set per istanziare un oggetto che può essere creato all'esterno per cui non è stata fornita una libreria di oggetti. (Per impostare una variabile oggetto a un'istanza esistente di un oggetto si usano Set e GetObject.)

Referenziare le applicazioni di Office 97

Per includere la libreria degli oggetti di Office 97 nella vostra applicazione Visual Basic, in modo che possa utilizzare la barra degli strumenti e il Raccoglitore, verificate che nella finestra di dialogo *References* (sotto *Project}* sia selezionata la Microsoft Office 8.0 Object Library.

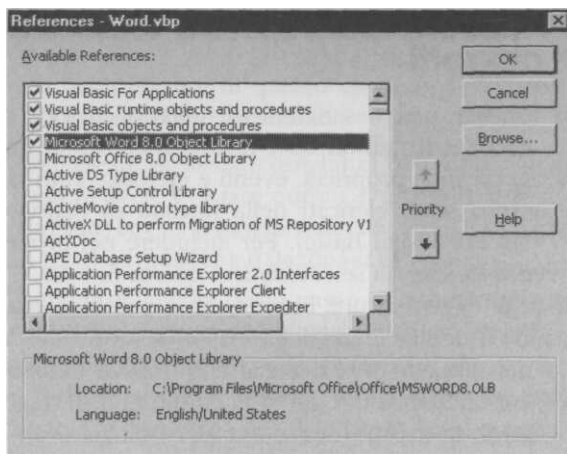
Allo stesso modo potete abilitare i riferimenti ad applicazioni specifiche di Office 97: per esempio, per includere la libreria degli oggetti di Word 8.0 è necessario selezionare la Microsoft Word 8.0 Object Library. La Figura 22.1 mostra la finestra *Project References,* in cui sono selezionate sia la libreria di oggetti di Office 97 che quella di Word.

Una volta che avete incluso il riferimento alla libreria degli oggetti, l'Object Browser permette di esaminare gli oggetti e i membri esposti dall'applicazione che ha generato la libreria.

Uso di metodi e proprietà degli oggetti

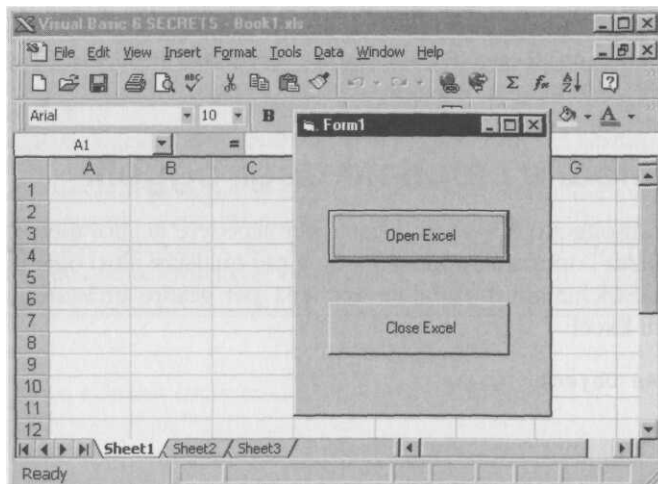
Una volta che un oggetto è stato istanziato, per accedere ai suoi metodi ed alle sue proprietà si utilizza l'operatore punto (.), come per qualsiasi altro oggetto. Per esempio, si potrebbe dichiarare la variabile appXcel per gestire un'istanza dell'oggetto Application di Excel:

```
Dim appXcel As Object
```

Figura 22.1 *La finestra di dialogo Project References permette di includere nei progetti Visual Basic riferimenti ad oggetti di applicazioni esterne.*

Poi si potrebbe creare un'istanza dell'oggetto applicazione Excel, farle aprire un file, e modificarne la didascalia (si veda il Listato 22.1 e la Figura 22.2).

Listato 22.1 *Creazione e controllo di un 'istanza di Excel.Application.*

```
Private Sub cmdOpen_Click()
   Set appXcel = CreateObject("Excel.Application")
   appXcel.Workbooks.Open
      filename:="H:\VB6Secrets\Ch22\SourceCode\book1.xls"
      'Change file location as needed
   appXcel.Caption = "Visual Basic 6 SECRETS"
   appXcel.Visible = True
End Sub
```

Figura 22.2 *L'oggetto può essere controllato attraverso i suoi membri (in questo caso viene aperto un file in Excel).*

Oltre alla proprietà Caption, l'applicazione espone il metodo Quit. Si potrebbe chiamarlo e quindi distruggere l'istanza dell'oggetto applicazione Excel:

```
Private Sub cmdClose_Click()
    appXcel.Quit
    Set appXcel = Nothing
End Sub
```

Come avrete notato, il processo non ha niente di spaventosamente complicato. Tuttavia spesso i problemi nascono perché le gerarchle degli oggetti hanno nomi poco chiari e non sono ben documentate: per questo motivo l'automazione di applicazioni esterne può richiedere un buon numero di tentativi e fallimenti.

Visual Basic for Applications

Le principali applicazioni che compongono Office 97, tra cui Access, Excel, e Word, utilizzano Visual Basic for Applications (VBA) come linguaggio comune per macro e script. VBA, nella versione Office 97, è diventato un vero e proprio ambiente di sviluppo, e decisamente potente: non è più il linguaggio delle macro dei vostri padri.

Microsoft ha rilasciato VBA in licenza a numerose terze parti, il che significa che VBA si avvia a diventare il linguaggio di scripting standard per Windows. Esiste anche una versione ancora più "leggera" di Visual Basic, Visual Basic Scripting Edition (VBScript), orientata principalmente alle applicazioni Web.

VBA è un sottoinsieme di Visual Basic 6. Scoprirete che l'ambiente di sviluppo di VBA vi è molto familiare, sia dal punto di vista concettuale che da quello estetico. La Figura 22.3 riproduce l'ambiente di sviluppo delle macro VBA disponibile in Word 8.

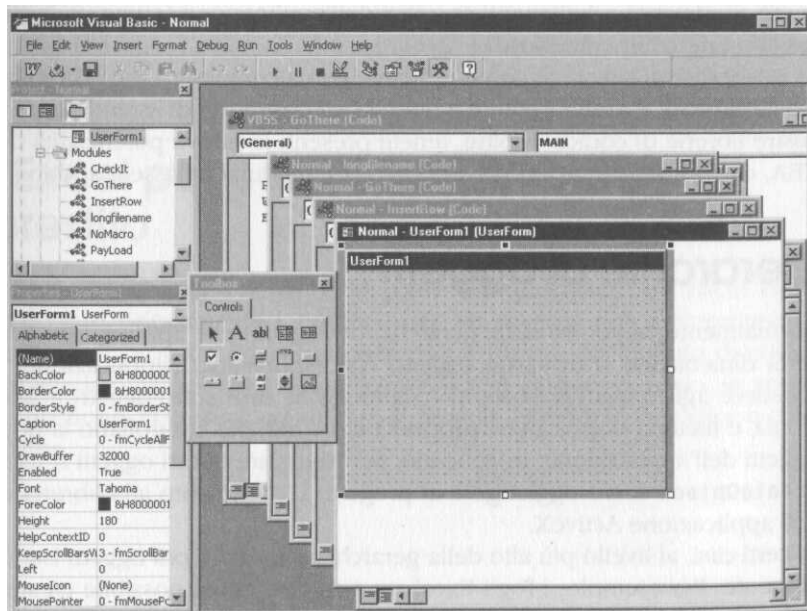
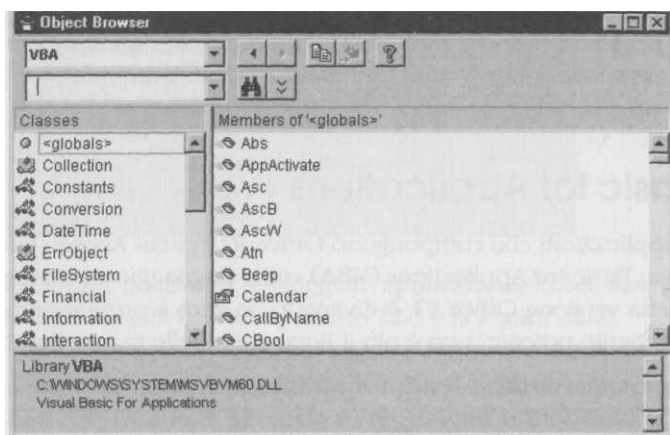
Figura 22.3 *L'ambiente di sviluppo di Visual Basic for Applications (VBA) risulterà immediatamente familiare a quanti programmano in Visual Basic.*

L'ambiente VBA, nella versione per le applicazioni di Office 97, permette di aggiungere form, moduli di codice e di classe, ma non moduli UserControl. Le finestre *Project Explorer, Toolbox* e *Properties* funzionano esattamente come nella versione "completa" di Visual Basic. Per rendervi conto esattamente di quali parti di Visual Basic siano incluse nel sottoinsieme VBA, potete aprire l'Object Browser in VB6 e selezionare VBA, come si vede nella Figura 22.4.

Figura 22.4
L'ObjectBrowser permette di visualizzare i membri della libreria VBA.

I membri elencati dall'Object Browser come componenti di VBA costituiscono la base del linguaggio di Visual Basic for Applications 6.0 (all'interno della libreria Vbvm6O.Dll). Il codice scritto in VB6 è portabile in VBA, almeno finché non fa riferimento ad oggetti esterni alla libreria del linguaggio VBA. È vero anche l'inverso: il codice scritto negli ambienti VBA può essere portato in Visual Basic 6, purché non faccia riferimento ad elementi specifici dell'applicazione (come un foglio di calcolo Excel o un documento Word).
La possibilità di saltare avanti e indietro (con alcuni limiti) tra VBA e il "fratellone", Visual Basic 6, è molto potente e dovrebbe consentirvi di estendere gli utilizzi delle vostre librerie di codice. Inoltre, tenete presenti le nuove potenzialità dell'ambiente VBA, che adesso mette a disposizione form, moduli di classe e add-in.

Gerarchie di oggetti

Normalmente, al vertice della gerarchia di oggetti delle applicazioni ActiveX di una certa dimensione si trova un oggetto che rappresenta l'applicazione, permette di accedere agli oggetti sottostanti (o almeno ai suoi sotto-oggetti e alle sue collezioni), e mette a disposizione proprietà e metodi che controllano la popolazione di oggetti dell'applicazione. In generale, per istanziare questi oggetti si usa la funzione CreateObject senza aggiungere al progetto il riferimento alla libreria degli oggetti dell'applicazione ActiveX.
In certi casi, al livello più alto della gerarchia si trovano più oggetti istanziabili esternamente. Per esempio, i fogli Excel sono oggetti che è possibile creare e utilizzare

direttramante. Di solito, l'oggetto base ha lo stesso nome dell'applicazione, per esempio Excel.Application o Word.Application. Questa non è però una regola: per esernpio, l'oggetto base in un'istanza dell'ambiente Visual Basic è VBIDE.VBE. l'accesso ai sotto-oggetti avviene per livelli successivi: prima di poter creare ed utilizzare un sotto-oggetto è necessario creare l'oggetto di livello superiore. Spesso i sotto-oggetti sono in realtà collezioni di oggetti: per raggiungere l'oggetto specifico che si vuole manipolare si usano in generale i metodi esposti dalla collezione stessa Per esempio, il codice riportato nel Listato 22.1 istanzia un oggetto applicazione Excel e lo memorizza nella variabile appXcel, quindi chiama il metodo Open della collezione WorkBooks dell'applicazione Excel:

```
appXcel.Workbooks.Open
```

In modo del tutto analogo, potremmo creare un foglio Excel impostando la variabile X, quindi usare il metodo Add della collezione Buttons di X per aggiungervi un pulsante:

```
Dim X as Object
Set X = CreateObject ("Excel.Sheet")
Dim Y as Excel.Button
Set Y = X.Buttons.Add (44,100,100,44)
Y.Caption = "My Button"
```

Per maggiori informazioni sull'uso delle collezioni, si veda il Capitolo 14.

Per un esempio i gerarchia di oggetti, si veda il Capitolo 29, chepresenta una discussione dettagliata di VBIDE.VBE.

Il modo più semplice per stabilire la struttura della gerarchia di oggetti di un'applicazione di Office è aprire la relativa libreria nell'Object Browser e premere il pulsante della guida (visualizzato come punto interrogativo) nell'angolo in alto a destra della finestra del browser. La Figura 22.5 riproduce la schermata della gerarchia di oggetti di Excel.

Uso di Excel per calcolare i rimborsi di un prestito

Questo esempio, disponibile sul CD-ROM come LoanCalc.Vbp, usa Excel come server per calcolare velocemente i rimborsi per un prestito con ammortamento (vedi Figura 22.6). Il progetto LoanCalc prevede un form e un modulo di codice .Bas che contiene la funzione che interagisce con Excel: in questo modo, se volete riutilizzare la funzione, vi basterà aggiungere al progetto il modulo di codice e chiamare la funzione.

Figura 22.5

Il tasto Help dell'Object Browser fornisce informazioni sulla gerarchia di oggetti di un 'applicazione (in questo caso l'oggetto Application di Excel).

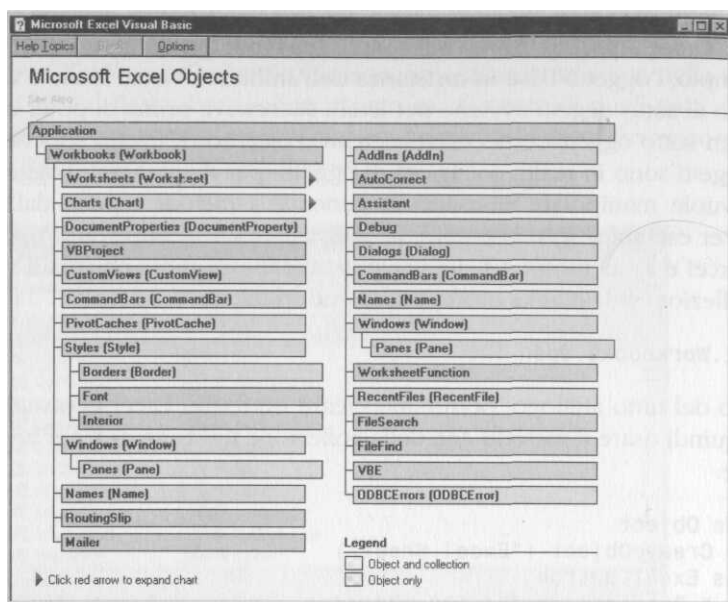
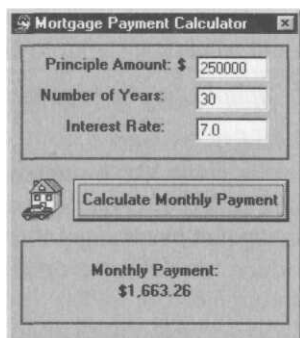



Figura 22.6

È possibile usare Excel come componente di automazione ActiveX per svolgere una serie di attività di supporto a un 'applicazione Visual Basic.

Si usa una funzione perché qualsiasi chiamata a oggetti e metodi di un server di automazione ActiveX potrebbe fallire. Esempio banale: il componente ActiveX potrebbe non essere correttamente installato, o non essere del tutto presente sulla macchina di destinazione. Visto che, notoriamente, i componenti ActiveX a volte falliscono, è buona norma incapsulare le chiamate ai server ActiveX in funzioni che restituiscono un valore che indica se l'operazione è andata a buonfine. Per esempio:

```
Public Function DoActiveXServerCall(FormalParameters) as Boolean
    DoActiveXServerCall = False
    'Controlla la validità di
    'Se non va bene, esce dalla funzione
    'Crea un'istanza del server

    'Esegue operazioni con il server

    'Imposta l'istanza del server = Nothing
```

```
    DoActiveXServerCall = True
End Function
```

Questo permette di chiamare la funzione e verificarne l'esito in una sola istruzione, I form che chiamano in questo modo funzioni presenti in moduli di codice sono a volte detti "form di interfaccia".

```
Private Procedure cmdDoSomething_Click()
    If Not DoActiveXServerCall(Arguments) Then
        MsgBox "La chiamata di automazione ActiveX è fallita!"
    Else
        'Usa i valori di ritorno ActiveX per aggiornare il form
    End If
End Procedure
```

Questo approccio non solo consente che l'applicazione (e l'utente) vengano avvertiti se un server non opera correttamente, ma permette di proseguire con le normali attività solo se la chiamata al server ha avuto buon esito.
Questo è il codice del form di interfaccia del progetto (vedi Figura 22.7) che chiama l'oggetto server Excel e, se la chiamata ha successo, popola il form utilizzando il risultato:

```
Private Sub cmdCalc_Click()
Dim Payment As Currency
    If Not CalcPay(CSng(txtAmount), Val(txtYears), _
        CSng(txtInterest), Payment) Then
        MsgBox "Vai a casa! Non possiamo calcolarlo!", _
            vbCritical, ProgTitle
    Else 'Calcolo andato a buon fine!
        Image1.Visible = True
        Label3 = Format(Payment, "$#,##0.00;($#,##0.00)")
    End If
End Sub
```

L'evento clic chiama la funzione CalcPay, che a sua volta chiama il server. Se Calc - Pay fallisce, viene mostrato un messaggio di avvertimento; se ha successo, il risultato viene riportato in Label3, formattato correttamente con la funzione Format. A proposito, se si imposta a True la proprietà Visible di Imagel verrà visualizzata l'icona della casa con l'auto (vedi Figura 22.6). ProgTitle è una costante globale definita nel modulo di codice, utilizzata per definire il titolo del progetto.
I parametri passati a CalcPay sono i dati forniti dall'utente, convertiti utilizzando le funzioni predefinite CSng e Val di VB. Payment è una variabile di tipo Currency utilizzata per restituire il risultato del calcolo della funzione. Questo "valore di ritorno" poteva essere combinato con il valore restituito dalla funzione per eliminare un parametro (per esempio restituendo O o un numero negativo in caso di errore), ma in generale è meglio evitare di attribuire più di un ruolo ai parametri e al valore restituito dalle funzioni: meglio usare una variabile per restituire un valore e un'altra o il risultato della funzione) per indicare l'esito dell'operazione. Il Listato 22.2 contiene il modulo di codice completo, compresa la funzione CalcPay:

Listato 22.2 *Uso di un oggetto Excel per calcolare l'ammortamento di un prestito.*

```
Option Explicit
Public Const ProgTitle = "Calcolo restituzione mutuo"

Public Function CalcPay(Amount As Currency, Years As Integer, _
   Interest As Single, Payment As Currency) As Boolean
   On Error GoTo HandleError
   Dim xlApp As Object
   ' Il tipo di oggetto Excel
   Const hdExcelObject = "Excel.Application"
   Screen.MousePointer = vbHourglass
   CalcPay = False
   ' istanzia l'oggetto applicazione Excel
   ' che eseguirà il calcolo!
   Set xlApp = CreateObject(hdExcelObject)
   ' Chiama il metodo Print di Excel
   Payment = xlApp.Pmt((Interest / 100) / 12, Years * 12, _
      -1 * Amount)
   xlApp.Quit
   Set xlApp = Nothing
   CalcPay = True
   Screen.MousePointer = vbDefault
   Exit Function
HandleError:
   'Stabilisce quale errore si sia verificato
   Select Case Err.Number
     Case 429
       MsgBox "Impossibile creare oggetto Automazione OLE" + _
          vbCrLf + "Verifica che Excel (v. 5.0 o superiore)" + _
          " sia stato installato correttamente.", vbCritical, ProgTitle
     Case Else
       MsgBox "Error #" + Str(Err.Number) + ": " + _
          Err.Description + ".", vbCritical, ProgTitle
   End Select
   Screen.MousePointer = vbDefault
End Function
```

Come potete vedere, una volta che il server di automazione Excel è stato avviato, lo scopo della funzione viene raggiunto con una semplice riga di codice.

Il primo passo è la dichiarazione di xlApp come oggetto. Visto che la dichiarazione è locale al contesto della funzione CalcPay, l'istanza del server che viene creata deve essere distrutta quando l'esecuzione esce dalla funzione. Per ribadire il concetto (ne vedremo una dimostrazione nell'esempio sull'intestazione di un documento Word più avanti in questo capitolo), se volete che un oggetto server resti disponibile al termine della funzione dovete dichiarare l'oggetto che rappresenta l'istanza del server a livello globale e non a livello di singola procedura.

In questo caso, tanto per andare sul sicuro, prima di uscire la funzione chiude l'istanza dell'oggetto Excel chiamandone il metodo Quit e impostando la variabile oggetto a Nothing (una sana abitudine per essere sicuri che tutta la memoria allocata venga rilasciata):

```
Set xlApp = Nothing
```

Questa è l'istruzione che crea l'istanza dell'oggetto OLE Server Excel:

```
Set xlApp = CreateObject(hdExcelObject)
```

Notate che hdExcelObject era stata definita come:

```
Const hdExcelObject = "Excel.Application"
```

Resta solo da svolgere il calcolo vero e proprio, utilizzando il metodo Pmt di xlApp: questa funzione finanziaria dei fogli Excel restituisce la rendita mensile sulla base di pagamenti e tassi di interesse costanti. Per utilizzare la funzione nel caso di un prestito, invece che di una rendita, basta invertire il segno dell'importo totale:

```
Payment = xlApp.Pmt((Interest / 100) / 12, Years * 12, _
      -1 * Amount)
```

Ovviamente, lo stesso calcolo si potrebbe effettuare direttamente in Visual Basic, senza passare da Excel, ricavando la formula da quella generale per gli interessi annuali composti:

```
Total_Sum = Principle * (1 + Interest_Rate / 100)Years
```

Ma ne varrebbe la pena? È importante prevedere una gestione degli errori che almeno mostri un messaggio di descrizione degli errori interni: se qualcosa non funziona nel processo di chiamata di un server di automazione, avrete bisogno di tutti gli indizi disponibili per scoprire la causa del problema.

Inserimento di un controllo Excel

A proposito, immaginiamo di voler elaborare un poco la nostra applicazione personalizzata e visualizzare un foglio di calcolo con un piano dei pagamenti annuali che riporti importi e interessi. Certo, non è complicato farlo in Excel. Potremmo allora usare un controllo OLE container per visualizzare i fogli di calcolo risultanti (si veda il Capitolo 21).

Un altro passo avanti sarebbe aggiungere alla Toolbox Excel come oggetto inserite, utilizzando la scheda *Insertable Objects* della finestra di dialogo *Components,* come si vede nella Figura 22.7: in questo modo nella casella degli strumenti sarà disponibile un oggetto Excel Sheet, come si vede nella Figura 22.8.

Figura 22.7
Per poter aggiungere ad un form un oggetto inseribile è prima necessario abilitarlo utilizzando la scheda Insertable Objects della finestra di dialogo Components.

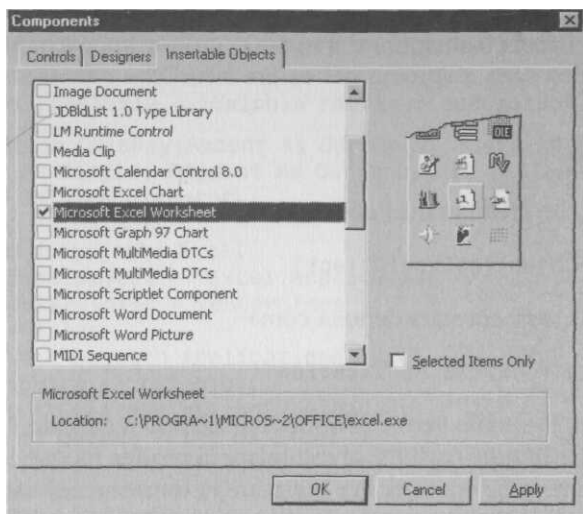



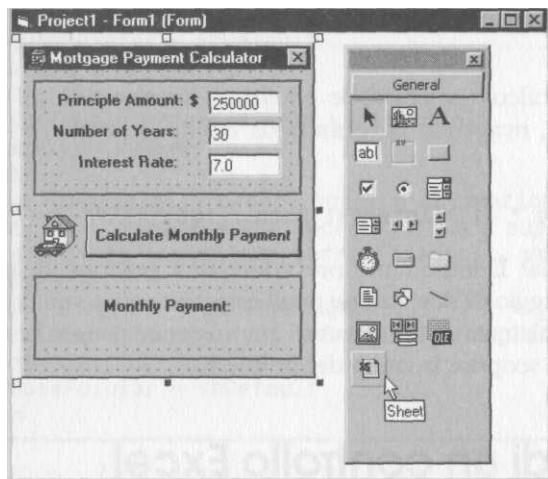
Figura 22.8
Gli oggetti ***inseribili*** *selezionati nella finestra di dialogo Components compaiono nella casella degli strumenti.*

A questo punto possiamo aggiungere al form un'istanza di Excel (vedi Figura 22.9): il funzionamento sarà praticamente lo stesso che avremmo ottenuto incorporando l'oggetto in un controllo contenitore, ma in questo caso non avremo a disposizione le proprietà, i metodi e gli eventi del contenitore. Il foglio di lavoro prevede l'attivazione in loco, ma per quasi tutte le altre manipolazioni dovremo usare l'oggetto esposto da Excel.

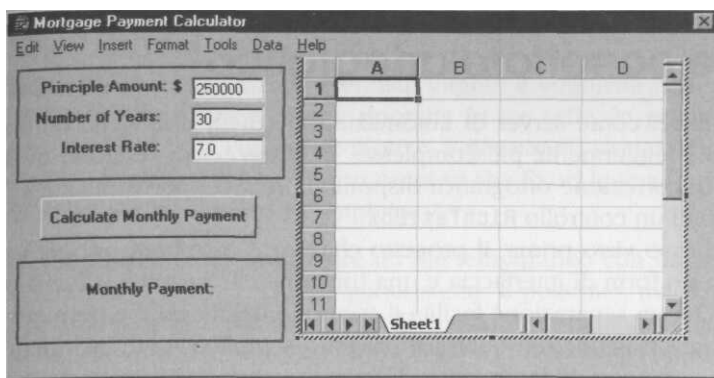
Figura 22.9 *Il foglio di lavoro di Exel può essere aggiunto come qualsiasi altro controllo*

Come si vede dalla finestra *Properties* riprodotta in Figura 22.10, l'elenco delle proprietà dell'oggetto Excel inserito disponibili in fase di progettazione è abbastanza limitato: la maggior parte delle proprietà è impostata in Excel, non nella modalità di progettazione di VB.

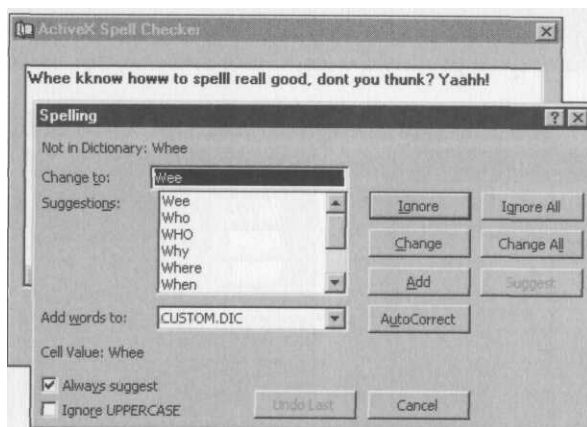
Figura 22.10 *La finestra Properties di Visual Basic non contiene molte delle proprietà dell'oggetto Excel inserito.*

Per accedere alle impostazioni delle proprietà dell'oggetto Excel inserito, fate clic col pulsante di destra sul controllo e selezionate Edit *dal menu che compare: l'oggetto Excel verrà apeno in modifica. È interessante notare come in questo modo i menu di Excel si integrino (il più delle volte sostituendoli) con quelli di Visual Basic.*

Uso di un server Excel come correttore ortografico

L'uso di Excel come server di automazione ActiveX che viene fatto nel progett Spell.Vbp è leggermente più complesso: in questo caso vengono utilizzate le fun zionalità di correzione ortografica disponibili in Excel per verificare e correggere '] contenutodiuncontrolloRichTextBox.

Come nel caso visto prima, il progetto che usa il correttore ortografico è diviso m due parti: un form di interfaccia e una funzione che gestisce il servizio OLE vero e proprio. Questa separazione facilita il controllo degli errori, e permette di utilizzare il correttore ortografico in qualsiasi progetto semplicemente includendo il modulo di codice e chiamando la funzione. Il form di interfaccia è piuttosto semplice-, comprende un controllo RichTextBox ed un pulsante (Figura 22.11).

Figura22.11
Èpossibileusare Excel come server di automazione ActiveX per verificare e correggere l'ortografia del testo inserito in un controllo Visual Basic.

Ecco il codice completo per il form di interfaccia:

```
Option  Explicit

Private Sub cmdCheck_Click()
   If  Not  SpellCheck(Me.RichTextBox1.)  Then
      MsgBox "No, oggi non posso fare controlli!", _
      vbExclamation, "No OLE"
   End If
   Me.SetFocus
End Sub
Private Sub Form_Unload(Cancel As Integer)
   End
End Sub
```

Il codice chiama la funzione SpellCheck passando come argomento il controllo RichTextBox, e fa in modo che il focus ritorni al form di interfaccia una volta che la funzione ritorna il controllo. La funzione SpellCheck richiede un utilizzo di Excel leggermente più sofisticato rispetto all'esempio del prestito, sostanzialmente per due motivi:

Il testo contenuto nel controllo VB deve essere salvato su disco ed essere letto in Excel: ogni parola viene inserita in una cella.

Excel non gestisce correttamente eventuali virgole e virgolette nelle celle: la stringa "Hello, my dearest" viene riportata in tre celle, la prima delle quali contiene "Hello",".". Per questo motivo, è necessario sostituire virgole e virgolette nel testo da controllare con caratteri che Excel ignora, e ripristinarle una volta che il controllo è completato.

Per salvare un file su disco, bisogna creare un file temporaneo con un nome completo che non entri in conflitto con i file già esistenti. È possibile, ma decisamente macchinoso, costruire da sé un nome di file di questo tipo, creando una directory temporanea e un nuovo file in essa. Ma la funzione API GetTempFileName di Windows permette di ottenere tutto questo in un solo colpo.

GetTempFileName *crea un file temporaneo in base ai tre parametri che vengono passati: la directory in cui crearlo, un prefisso di tre lettere per il nome del file e un intero senza segno da cui viene derivata una stringa esadecimale che completa il nome del file. Il quarto argomento è un puntatore al nome di file; il valore restituito dalla funzione non ci serve.*

Se come primo argomento si passa un punto ("."), il file viene creato nella directory corrente, il che normalmente è accettabile. Il prefisso di tre lettere può essere qualsiasi, mentre l'intero senza segno dovrebbe essere sempre 0: in questo modo la stringa esadecimale sarà determinata sulla base dell'orologio di sistema.
Il Listato 22.3 presenta la dichiarazione di GetTempFileName e la funzione che ne incapsula le operazioni. Alla funzione sono sufficienti i primi due parametri per restituire il nome unico per un file temporaneo. (Se il primo parametro è ".", il file temporaneo viene creato nella directory corrente.)

Listato 22.3 *Come ottenere il nome di un file temporaneo.*

```
 'usato per creare un file temporaneo
 'che possa essere aperto nell'oggetto Excel
Declare Function GetTempFileName Lib "kernel32" Alias _
   "GetTempFileNameA" (ByVal lpszPath As String, -
   ByVal lpPrefixString As String, ByVal wUnique As Long, _
   ByVal lpTempFileName As String) As Long

Public Function GetTempFile(Directory As String, _
   Prefix As String) As String
   Dim FileName As String
   FileName = String(256, 0)
     'prende un nome di file temporaneo
   Call GetTempFileName(Directory, Prefix, 0, FileName)
     'elimina il terminatone nullo
   FileName = Left(FileName, InStr(FileName, Chr(0)) - 1)
     'restituisce i risultati
   GetTempFile = FileName
   Function
```

Ecco come avviene la chiamata alla funzione da parte di SpellCheck:

```
' prende un nome di file temporaneo nella directory corrente
FileName = GetTempFile(".", "OLE")
```

Come si vede nel Listato 22.4, il problema con la gestione di virgole e virgolette da parte di Excel viene gestito da una funzione di manipolazione della stringa che permette prima di sostituire tali caratteri con altri che Excel ignora, e poi di ripristinarli

Listato 22.4 *Sostituzione dei caratteri.*

```
Private Function StripText(InChar As Integer, _
   OutChar As Integer, InText As String) As String
   ' Sostituisce InChar con OutChar in InText
   Dim StartPos As Integer
   Dim FoundPos As Integer
   StartPos = 1
   FoundPos = InStr(StartPos, InText, Chr(InChar))
   While FoundPos > 0
      Mid(InText, FoundPos, 2) = Chr(OutChar)
      StartPos = FoundPos + 1
      FoundPos = InStr(StartPos, InText, Chr(InChar))
   Wend
   StripText = InText
End Function
```

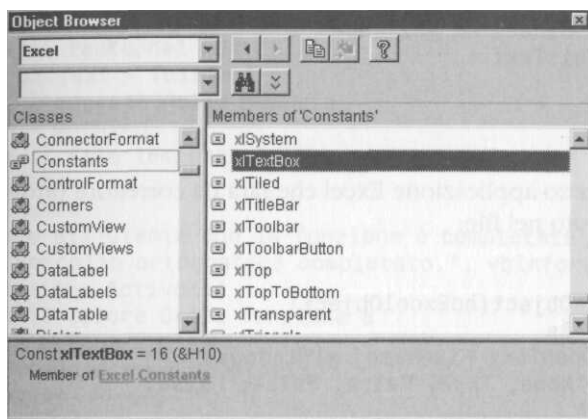
Come si vede dal codice, la funzione StripText accetta come parametri il carattere da sostituire ed il sostituto come codice ASCII. Per prima cosa, la funzione SpellCheck dichiara le variabili interne, quelle utilizzate per dichiarare gli oggetti Excel, le costanti di Excel e le costanti utilizzate per la funzione StripText. Il tutto è riportato nel Listato 22.5.

Listato 22.5 *Usare un oggetto Excel come correttore ortografico,*

```
Public Function SpellCheck(ThisControl As Contro!) As Boolean
   On Error GoTo HandleError
   Dim xlApp As Object
   Dim xlWorkBook As Object
   Dim xlWorkSheet As Object
   Dim FileName As String
   Dim FileName2 As String
   Dim ThisText As String
   Dim FileNum As Integer
   Const xlTextPrinter = 36
   Const xlWindows = 2
   Const xlDelimited = 1
   Const xlNone = -4142
   Const xlText = -4158
   Const xlTextWindows = 20

    'II tipo di oggetto Excel
   Const hdExcelObject = "EXCEL.APPLICATION"
```

```
'costanti per sistemare il testo - codici ASCII
Const hdCommaText = 44    'Virgola
Const hdCommaSub = 147    'Non usato
Const hdQuoteText = 34    'Apostrofo
Const hdQuoteSub = 148    'Non usato
Const hdTabText =9        'Tab
Const hdSpaceText = 32    'Spazio
```

Il modo più semplice per ricavare il valore delle costanti per un server ActiveX esterno è utilizzare l'Object Browser, come si vede in Figura 22.12. Verificate che la relativa libreria di oggetti sia stata selezionata nella finestra di dialogo References *(accessibile dal menu* Tools).

Figura 22.12
L'Object Browser è lo strumento più semplice per trovare il valore delle costanti per server di automazione ActiveX.

La funzione SpellCheck procede impostando inizialmente il proprio risultato a False ed utilizzando la struttura TypeOf. . . Is per verificare che il controllo passato come parametro sia una RichTextBox o una casella di testo:

```
SpellCheck = False
If Not TypeOf ThisControl Is RichTextBox And _
   Not TypeOf ThisControl Is TextBox Then
   MsgBox "Errore interno: " + vbCrLf + _
      "Il controllo che stai versificando non è una casella di testo" _
      + vbCrLf + " né una RichTextBox!" + vbCrLf + _
      "Il controllo ortografico non funziona.", vbCritical, "OLE!"
   Exit Function
End if
```

Ora, completati i preliminari, passiamo al lavoro vero e proprio. Il cursore vien impostato con l'icona della clessidra, il testo del controllo viene letto in una varia bile interna, e vengono sostituite virgole e virgolette:

```
Screen.MousePointer = vbHourGlass
ThisText = ThisControl.Text
ThisText = StripText(hdCommaText, hdCommaSub, ThisText)
ThisText = StripText(hdQuoteText, hdQuoteSub, ThisText)
```

Si ottiene il nome di un file temporaneo nella directory corrente, e si salva il testo-

```
FileName = GetTempFile(".", "OLE")
FileNum = FreeFile
Open FileName For Binary As #FileNum
Put #FileNum, , ThisText
'chiude il file
dose #FileNum
```

Quindi, si crea l'oggetto applicazione Excel che farà da correttore ortografico e gli si fa aprire il testo salvato nel file:

```
Set xlApp = CreateObject(hdExcelObject)
'apre il nostro file
xlApp.Workbooks.OpenText FileName, xlWindows, 1, _
   xlDelimited, xlNone, True, False, False, False, _
   True, False, ""
'prende il foglio attivo
Set xlWorkSheet = xlApp.ActiveSheet
'prende la cartella di lavoro attiva
Set xlWorkBook = xlApp.ActiveWorkbook
```

Ecco l'istruzione che richiama il correttore ortografico:

```
xlWorkSheet.CheckSpelling
```

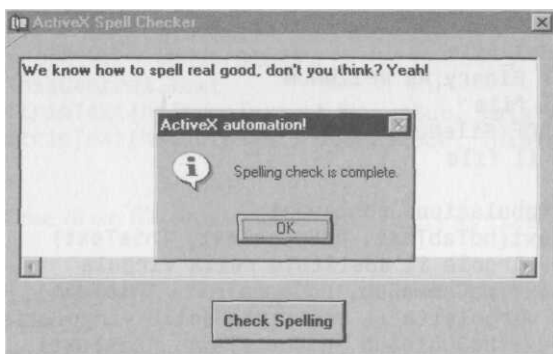
Non resta che ripulire:

```
 'prende un secondo nome di file
FileName2 = GetTempFile(".", "OLE")
 'Lo cancella
Kill FileName2
 ' ...ma usa ancora il nome salva il foglio
xlWorkSheet.SaveAs FileName2, xlTextWindows
 ' imposta la proprietà Saved perché Excel non ci chieda di salvare
xlWorkBook.Saved = True
```

```
      ' esce da Excel
   xlApp.Quit

      ' prende un numero di file libero
   FileNum = FreeFile
   ' apre il backup del file
   Open FileName2 Por Binary As #FileNum
   ' legge i dati nel file
   ThisText = Input(LOF(FileNum), #FileNum)
   ' chiude di nuovo il file
   Close #FileNum
   ' sostituisce le tabulazioni con spazi
   ThisText = StripText(hdTabText, hdSpaceText, ThisText)
   ' sostituisce con virgole il sostituto della virgola
   ThisText = StripText(hdCommaSub, hdCommaText, ThisText)
   ' sostituisce con virgolette il sostituto delle virgolette
   ThisText = StripText(hdQuoteSub, hdQuoteText, ThisText)
   ' rimette il testo nel controllo
   ThisControl.Text = ThisText
   ' elimina l'oggetto dalla memoria
   Set xlApp = Nothing
   ' elimina il file temporaneo
   Kill FileName
   Kill FileName2
   ' fa sapere all'utente che la funzione è completata
   MsgBox "Controllo ortografico completato.", vbInformation, _
      "Automazione ActiveX!"
   ' Imposta il valore della funzione a
   SpellCheck = True
   Screen.MousePointer = vbDefault
   Exit Function
HandleError:
   ' stabilisce quale errore si sia versificato
   Select Case Err.Number
     Case 429
       MsgBox "Impossibile creare oggetto OLE con Excel." + _
         vbCrLf + "Verifica che Excel (v. 5.0 o superiore)" + _
         "sia stato installato.", _
         vbCritical, "OLE Error"
     Case Else
       MsgBox "Errore #" + Str(Err.Number) + ": " + _
         Err.Description + _
         ".", vbCritical, "Errore OLE"
   End Select
   Screen.MousePointer = vbDefault
End Function
```

Effettivamente, pare che sia necessario brigare parecchio perché Excel si comporti esaattamente come vogliamo. È vero, ma funziona, come dimostra la Figura 22.13- Ed è sempre meno faticoso che scriversi da zero un programma di aggiunta che faccia da correttore ortografico, e non c'è neanche bisogno di definire un dizionario, visto che Excel ha già il suo! Una volta che avete scritto la funzione SpellCheck, potete riutilizzarla in qualsiasi programma semplicemente aggiungendo al vostro progetto il modulo di codice e chiamando SpellCheck. (Come al solito, quando si

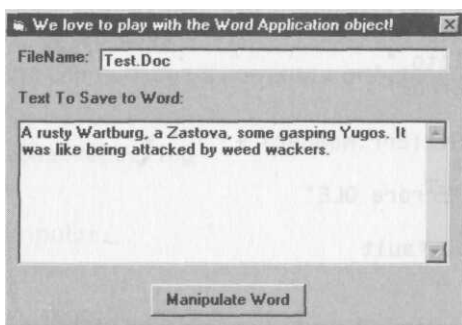
parla di server di automazione ActiveX, vale la condizione "a patto che il server s' stato correttamente installato sul sistema di destinazione".)

Figura 22.13 *Usando Excel come server OLE, è facile correggere gli errori di ortografia nel testo.*

Creazione e modifica di documenti Word

Per programmare Office 97 e Word 8 non è più necessario complicarsi la vita con Word Basic: Word 8 usa VBA come linguaggio nativo per le macro.

Come esempio, vedremo come utilizzare la gerarchia di oggetti Word .Application *per creare un documento per cui l'utente fornisce un nome. Il programma di esempio, disponibile sul CD-ROM come Word.Vbp, inserisce nel documento anche un 'intestazione, la data e il testo specificato dall'utente.*

Il form di interfaccia, riprodotto in Figura 22.14, presenta due caselle di testo che ricevono dall'utente la definizione del nome del file per il documento Word e il testo da inserire nel corpo del documento.

Figura 22.14 *L'oggetto Word.Application permette di manipolare documenti Word da applicazioni Visual Basic.*

L'oggetto Word.Basic

Word 7 esponeva un solo oggetto, Word.Basic, ma estremamente potente: i suoi metodi erano i comandi del linguaggio per macro WordBasic.
Con il superamento della versione 7 di Word e della controversa mutazione del linguaggio Basic che era Word Basic, Word 8 si è affiancato alle altre applicazioni Office imp lementando una struttura di oggetti applicativi standard. Questa gerarchia risulterà in qualche modo estranea a chi ha programmato utilizzando il vecchio modello di oggetti di Word, ma si avvicina molto al modello di gerarchia di oggetti generale, nel senso che è molto più robusta e *object-oriented* di qualsiasi incarnazione dell'oggetto Word.Basic.
Dovete comunque sapere che la maggior parte del codice scritto utilizzando il modello Word.Basic di Word 7 continua a funzionare con Word 8, perché l'oggetto Word. Basic è stato conservato per garantire la compatibilita con le versioni precedenti.

Ecco il codice per il form di interfaccia:

```
Option Explicit
Private Sub cmdSave_Click()
   If Not ManipulateWord(txtSaveText, txtFileName.Text) Then
     MsgBox "Impossibile manipolare", vbInformation, ProgTitle
   End If
End Sub

Private Sub Form_Load()
   Me.Caption = ProgTitle
End Sub

Private Sub Form_Unload(Cancel As Integer)
   End
End Sub
```

ProgTitle è una costante globale che riporta il nome dell'applicazione, ed è dichiarata nella sezione Declarations del modulo di codice. Anche l'oggetto applicazione Word è dichiarato come Public nello stesso punto, in modo che resti valido per la totalità del progetto e Word rimanga aperto per tutta la durata dell'operazione.

Se volete che Word venga chiuso al termine dell'esecuzione dellafunzione, basta eliminare la dichiarazione dell'oggetto come variabile pubblica, che lo rende globale rispetto alprogetto, e renderla locale rispetto allafunzione.

Ecco le dichiarazioni:

```
Option Explicit
   Public objWord As Object 'Si sposta nella funzione perché Word
                            'si chiuda dopo l'esecuzione
'nomeprogramma
     Public Const ProgTitle = "Ci piace giocare con Word" + _
    "Oggetto applicazione!"
```

Il Listato 22.6 riporta la funzione ManipulateWord.

Listato 22.6 *La funzione* `ManipulateWord.`

```
Public Function ManipulateWord(ThisControl As Control, _
   ThisFileName As String) As Boolean
   Dim objDoc As Object 'Variabile per conservare l'oggetto documento Word
   On Error GoTo HandleErr
   'imposta il valore iniziale
   ManipulateWord = False
   If Not TypeOf ThisControl Is RichTextBox And _
      Not TypeOf ThisControl Is TextBox Then
         MsgBox "Errore interno: " + vbCrLf + _
            "Il controllo che stai verificando" + _
            " non è una casella di testo " _
            + vbCrLf + "né una RichTextBox!" + vbCrLf + _
            "La funzione di manipolazione di Word non funzionerà.", _
            vbCritical, ProgTitle
         Exit Function
   End If
   If ThisFileName = "" Then   'La casella del nome file è vuota
       MsgBox "Devi: fornire un nome di file" + _
          " per poter salvare il testo in.", _
          vbCritical, ProgTitle
       Exit Function
   End If
   Screen.MousePointer = vbHourglass
   'apre Word
   Set objWord = GetObject(, "word.application")
   'apre un nuovo documento
   Set objDoc = objWord.Documents.Add
   Documents(objDoc).Activate
   ActiveWindow.WindowState = wdWindowStateMaximize
   'imposta le informazioni di intestazione
   If ActiveWindow.View.SplitSpecial <> wdPaneNone Then
      ActiveWindow.Panes(2) .Close
   End If
   If ActiveWindow.ActivePane.View.Type = wdNormalView _
      Or ActiveWindow.ActivePane.View.Type = wdOutlineView _
      Or ActiveWindow.ActivePane.View.Type = wdMasterView Then
         ActiveWindow.ActivePane.View.Type = wdPageView
   End If
   ActiveWindow.ActivePane.View.SeekView = _
      wdSeekCurrentPageHeader
   Selection.Font.Bold = True       'attiva il grassetto
   Selection.InsertAfter ProgTitle  'inserisce il nome del programma
   Selection.InsertAfter Chr$(9)    'tab una volta
   'Poi inserisce la data
   Selection.InsertAfter Formattate, "mmmm g, aaaa")
   Selection.Font.Bold = False      'disattiva il grassetto
   'Chiude il pannello dell'intestazione
   ActiveWindow.ActivePane.View.SeekView = wdSeekMainDocument

   Dim button As Long
   Selection.InsertAfter vbCrLf  'inserisce un a capo
   Selection.InsertAfter vbCrLf
   'mette il testo nel documento
```

```
    Selection.InsertAfter  ThisControl.Text
    'è andata a buon fine!
    ManipulateWord = True
    Screen.MousePointer= vbDefault
    Set objDoc = Nothing
    Exit Function
HandleErr:
    MsgBox "Errore #" + Str(Err.Number) + ": " + _
        Err.Description + ".", vbCritical, ProgTitle
    Screen.MousePointer = vbDefault
End Function
```

Quando lanciate la funzione ManipulateWord da VB, viene aperto un nuovo documento con il nome specificato, e vengono inseriti l'intestazione e il corpo del testo (Figura 22.15).

Notate che, per come è scritta, per operare correttamente la funzione ManipulateWord, *riportata nel Listato 22.6, richiede che Word sia già aperto.*

Modifica di un database Access

Il prossimo esempio, Access.Vbp, usa l'oggetto Access.Application e prevede due funzioni: GetReports restituisce i nomi dei report definiti in un database Access, e PrintReports stampa una copia di ciascuno. La Figura 22.16 riproduce l'interfaccia del progetto, in cui è stato caricato l'onnipresente database Northwind Trading Company. (Il database Northwind è uno degli esempi distribuiti con Access, e viene utilizzato per fare pratica e dimostrazioni. Come riporta la schermata del database, ogni riferimento a nomi, società e dati esistenti utilizzati in esempi ed illustrazioni è puramente casuale. Immagino che questo esaurisca ogni possibilità!)

Figura 22.15 *Da Visual Basic si può controllare qualsiasi elemento dell'aspetto e del contenuto dei documenti Word.*

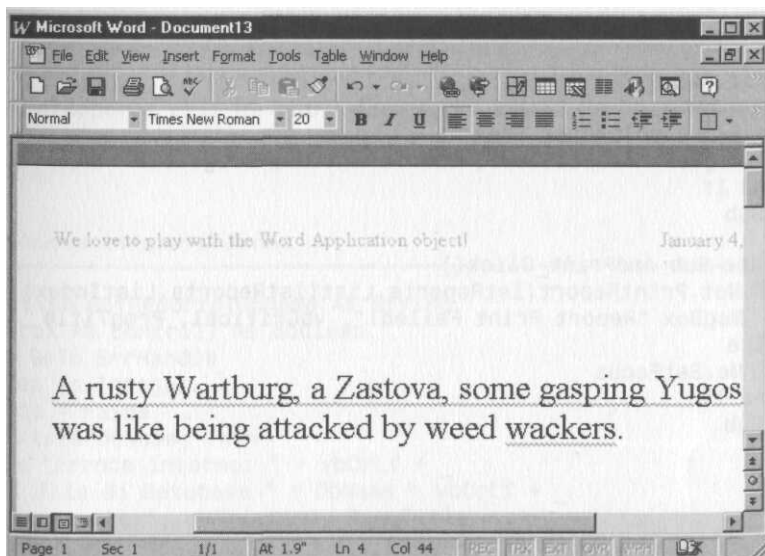
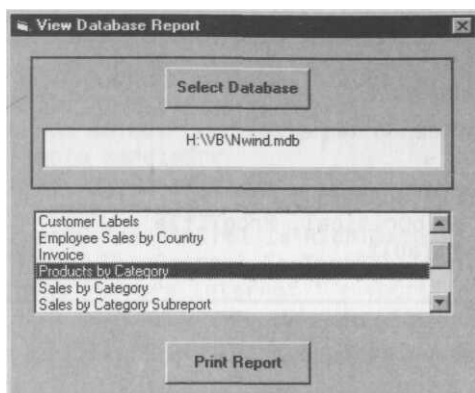



Figura 22.16 *Gli oggetti di automazione ActiveX esposti da Access permettono di elencare i report inclusi in un database Access.*

Il codice contenuto nel modulo di interfaccia del progetto segue lo standard per i client ActiveX, con in più la chiamata al controllo delle finestre di dialogo comuni per consentire all'utente di selezionare un database. (Per informazioni sull'uso del controllo dialogo comune, si veda il Capitolo 7.)

```
Private Sub cmdSelect_Click()
   On Error GoTo ErrHandle
   Dim OurFile As String
   CommonDialog1.CancelError = True
   CommonDialog1.DialogTitle = "Seleziona un database, per favore!"
   CommonDialog1.Filter = "Access Database Files" + _
      " (*.Mdb)|*.mdb|All Files (*.*)|*.*"
   CommonDialog1.Flags = cdlOFNFileMustExist
   CommonDialog1.ShowOpen
   txtDBName = CommonDialog1.filename
   On Error GoTo 0
   If Not GetReports(txtDBName, IstReports) Then
      MsgBox "Impossibile aprire il file di database selezionato|", _
          vbCritical, ProgTitle
   End If
   Exit Sub
ErrHandle:
   If Err = cdlCancel Then
      MsgBox "Annullato!", vbInformation, ProgTitle
   End If
End Sub

Private Sub cmdPrint_Click()
   If Not PrintReport(lstReports.List(lstReports.ListIndex)) Then
      MsgBox "Report Print Failed!", vbCritical, ProgTitle
   Else
      Me.SetFocus
   End If
End Sub
```

L'evento Unload per il modulo di interfaccia è leggermente più sofisticato rispetto a quelli visti finora, perché in questo caso ci aspettiamo che l'oggetto Access e il database resstino aperti dopo la selezione del database, e fino alla distruzione del form:

```
Private Sub Form_Unload(Cancel As Integer)
If Not objAccess Is Nothing Then
    objAccess.Quit
    Set dbs = Nothing
    Set objAccess = Nothing
End If
End
End Sub
```

Nel modulo di codice, l'oggetto Excel e il relativo database sono dichiarati globali:

```
Option Explicit
Public Const ProgTitle = "Visualizzazione report di database"
Public objAccess As Object
Public dbs As Object
```

La funzione GetReports usa la funzione Exists (descritta nel Capitolo 16) per fare almeno una minima verifica sul presunto file del database passato alla funzione. La funzione Exists permette almeno di verificare se il file esista (ma, ovviamente, non di controllare se si tratti effettivamente di un file .Mdb contenente un database Access). Questa è la funzione Exists:

```
Public Function Exists(F As String) As Boolean
    Dim X As Long
    On Error Resume Next
    X = FileLen(F)
    If X Then
        Exists = True
    Else
        Exists = False
    End If
End Function
```

La funzione GetReports cicla fra tutti i documenti della collezione Reports del database Access e ne aggiunge i nomi al controllo casella di riepilogo, come si vede nel Listato 22.7:

Listato 22.7 *Ricerca dei report di Access.*

```
Public Function GetReports(DbName As String, _
ThisControl As Control) As Boolean
On Error GoTo ErrHandle
Dim IntRep As Integer
GetReports = False
 If Not Exists(DbName) Then
   MsgBox "Errore interno: " + vbCrLf + _
      "Il file di database " + DbName + vbCrLf + _
      "non esiste.", vbCritical, ProgTitle
   Exit Function
```

```
   End If
   If Not TypeOf ThisControl Is ListBox Then
      MsgBox "Errore interno: " + vbCrLf +
         "Il controllo non è una casella di riepilogo", vbCritical,ProgTitle
      Exit Function
   Else
      ThisControl.Clear
   End If
   Screen.MousePointer = vbHourglass
   Set objAccess = CreateObject( "Access.Application")
   With objAccess
      .OpenCurrentDatabase (DbName)
      Set dbs = .CurrentDb
      With dbs.Containers("Reports")
         For IntRep = 0 To .Documents.Count - 1
            If Left(.Documents(IntRep).Name, 4) <> "~TMP" Then
               ThisControl.AddItem .Documents(IntRep).Name
            End If
         Next IntRep
      End With
   End With
   GetReports = True
   Screen.MousePointer = vbDefault
   Exit Function
ErrHandle:
   MsgBox "Errore #" + Str(Err.Number) + ": " + _
      Err.Description+".", _
      vbCritical, ProgTitle
   Screen.MousePointer = vbDefault
End Function
```

La funzione `PrintReport` completa l'opera di `GetReports`. Sostanzialmente, una volta che il relativo database è aperto in Access, una riga di codice è sufficiente per stampare il report:

```
Public Function PrintReport(WhichReport) As Boolean
   On Error GoTo ErrHandle
   Screen.MousePointer = vbHourglass
   PrintReport = False
   If WhichReport = "" Then
      MsgBox "Errore interno: " + vbCrLf + _
         "Stringa report non passata.", vbCritical, ProgTitle
      Exit Function
   End If
   objAccess.DoCmd.OpenReport WhichReport ',acPreview per anteprima
   PrintReport = True
   Screen.MousePointer = vbDefault
   Exit Function
ErrHandle:
   MsgBox "Errore #" + Str(Err.Number) + ": " + _
      Err.Description + ".", _
      vbCritical, ProgTitle
   Screen.MousePointer = vbDefault
End Function
```

Per avere un'anteprima invece di stampare il report basta aggiungere il parametro opzionale acPreview (una costante pari a 2) sulla riga di codice che stampa, dopo il nome del report:

```
objAccess.DoCmd.OpenReportWhichReport, acPreview
```

Il risultato dell'aggiunta della costante acPreview è visibile in Figura 22.17. Ancora una volta, la miglior fonte di informazione sul valore delle costanti di Access è l'Obiect Browser (ovviamente, la libreria di oggetti di Microsoft Access 8.0 deve essere tra i riferimenti attivi per il progetto).

L'oggetto DoCmd *di Access ha lo scopo di eseguire comandi: per l'elenco dei metodi supportati e la descrizione del loro funzionamento fate riferimento alla guida in linea di Access.*

Figura 22.17 *L'anteprima di stampa del report "Products By Company" del database di esempio Northwind Trading Company di Access.*

Riepilogo

In questo capitolo abbiamo visto come esaminare e controllare gli oggetti di un'applicazione server OLE da Visual Basic.

- Abbiamo definito le librerie di oggetti.
- Abbiamo visto come creare un'istanza di un oggetto.
- Abbiamo visto come consultare le gerarchie di oggetti.
- Abbiamo visto come stabilire l'ambito di validità dei riferimenti agli oggetti.
- Abbiamo visto come realizzare la gestione degli errori e restituire l'esito di una operazione in funzioni che utilizzano componenti ActiveX.

- Abbiamo visto come usare Excel come server di automazione ActiveX per calcolare l'ammortamento di un prestito.
- Abbiamo visto come usare Excel come server di automazione ActiveX per verificare l'ortografia del testo inserito dall'utente in controlli Visual Basic.
- Abbiamo visto la funzione dell'API GetTempFileName.
- Abbiamo visto come creare una funzione di incapsulamento per GetTemp-FileName.
- Abbiamo visto come usare una usare una generica funzione di analisi per sostituire all'interno di una stringa un carattere con un altro.
- Abbiamo visto come controllare Word 8 e l'oggetto Word.Application da VisualBasic.
- Abbiamo visto come usare Access come server di automazione ActiveX per stampare i report definiti in un database.

CREAZIONE DI APPLICAZIONI ACTIVEX

- Concetti fondamentali delle applicazioni ActiveX
- Tipi di applicazione ActiveX: server in-process e out-of-process
- La proprietà Instancing per i moduli di classe
- Creazione di un'applicazione ActiveX passo per passo
- Creazione di un modulo di classe per incapsulare altre funzioni
- Gestione degli errori con componenti ActiveX
- ActiveX e registrazioni
- Creazione di oggetti applicativi
- Creazione di gerarchie di oggetti
- Creazione di server in-process (DLL ActiveX)
- Vincoli sulle DLL in-process

In questo capitolo vedremo l'altro lato della questione. Nei Capitoli 21 e 22 abbiamo visto come controllare i server ActiveX e implementare le funzionalità OLE nei progetti. Questo capitolo dimostra come creare applicazioni ActiveX. Usando le tecniche presentate negli ultimi due capitoli, le applicazioni client (scritte da voi o da altri) possono accedere e utilizzare gli oggetti esposti dai vostri componenti ActiveX. Questo è un tema molto stimolante: è possibile scrivere componenti ActiveX per incapsulare del codice che può essere riutilizzato all'infinito da voi o da altri per estendere l'ambiente di Visual Basic, per suddividere un programma di grandi dimensioni in moduli più piccoli, utilizzabili come librerie da applicazioni client OLE come Excel o Word per Windows, o utilizzabili come strumenti di sviluppo.

Concetti fondamentali

A questo punto, i meccanismi di base di ActiveX e OLE dovrebbero risultarvi familiari. Tuttavia, l'argomento è abbastanza importante da meritare un riassunto dei concetti e dei termini principali.

L'oggetto precedentemente noto come Server OLE

I componenti ActiveX sono quelle che venivano chiamate applicazioni server OLP È importante capire che i termini *componente ActiveX* e *controllo ActiveX* non riferiscono allo stesso oggetto. I controlli ActiveX, infatti, sono implementati come componenti ActiveX: è un altro modo per dire che i controlli ActiveX sono un tipo di componente ActiveX.

I componenti ActiveX possono essere implementati come *server out-of-process* nel senso che girano in un proprio spazio di esecuzione: in questo caso generalmente vengono compilati come file eseguibili con estensione .Exe. In alternativa, possono essere compilati come libreria a collegamento dinamico *in-process,* normalmente generando un file .DLL. Pensate ai componenti ActiveX come all'"oggetto precedentemente noto come server OLE".

Microsoft crede che la terminologia ActiveX possa semplificare la discussione delle interazioni tra oggetti senza ricorrere all'inflazionata parola "server". E, come si dice in Microsoft, "se proprio volete, potete continuare ad usare la vecchia terminologia: basta pronunciare 'componente ActiveX' come 'server OLE'".

Personalmente, a volte trovo più semplice pensare in termini di componenti ActiveX, altre volte a client e server. In questo capitolo, cercherò di usare la terminologia in accordo con il contesto in cui ci troveremo ad operare.

Moduli di classe e ActiveX

In questo libro abbiamo già visto una serie di utilizzi notevoli per le applicazioni ActiveX. Per esempio, nel Capitolo 10, abbiamo visto come creare un'applicazione ActiveX che incapsuli molte delle API del Registro di configurazione.

Le aggiunte di Visual Basic sono applicazioni ActiveX che contengono procedure di moduli di classe specifiche, che vengono chiamate da un'istanza dell'ambiente VB operante come client OLE. Nel Capitolo 29, vedremo come creare un'applicazione ActiveX che è un'aggiunta che modifica la proprietà `BackColor` di tutti i controlli di un form caricato nell'ambiente VB.

In linea di principio, la creazione di un'applicazione ActiveX non presenta particolari complicazioni. Come minimo, un'applicazione ActiveX deve contenere un modulo di classe la cui proprietà `Instancing` sia impostata in modo da consentire la creazione degli oggetti basati sulla classe dall'esterno. La Tabella 23.1 mostra il significato dei sei possibili valori per la proprietà `Instancing` di un modulo di classe. Come potete vedere, la creazione dall'esterno è possibile se la proprietà `Instancing` di un modulo di classe che fa parte di un progetto ActiveX è impostata a un valore diverso da 1 `-Private` o `2-PublicNotCreatable`.

Tabella 23.1 *le impostazioni per la proprietà Instancing di un modulo di classe.*

Valore	Descrizione
1 - Private	Il modulo di classe è privato (locale al contesto) del progetto, e non può essere creato esternamente. Le altre applicazioni non possono accedere alle informazioni della libreria di tipi riguardanti la classe, e non la possono istanziare. Gli oggetti privati possono essere utilizzati solo all'interno dell'applicazione/componente.
2-PublicNotCreatable	Non può essere creato esternamente, ma può essere utilizzato dopo che è stato creato dalPapplicazione/componente.
3-SingleUse	Permette alle altre applicazioni di creare gli oggetti sulla base della classe, ma ogni oggetto della classe creato da un client lancia una nuova istanza del server.
4-GlobalSingleUse	Simile a SingleUse, ma le proprietà e i metodi della classe possono essere richiamati come semplici funzioni globali.
5 - Multiuse	Permette alle altre applicazioni di creare gli oggetti basati sulla classe. Un'istanza dell'applicazione può fornire un qualsiasi numero di oggetti creati in questa modalità, indipendentemente da quante applicazioni li richiedano. Il server viene avviato se quando l'oggetto viene creato non è già in esecuzione.
6- GlobalMultiUse	Simile a Multiuse, ma le proprietà ed i metodi della classe possono essere richiamati come semplici funzioni globali. Non è necessario creare prima esplicitamente un'istanza della classe, perché questo avviene automaticamente.

Tuttavia, dire che basta che un'applicazione ActiveX abbia un modulo di classe creabile dall'esterno è come dire che scrivere un'applicazione VB è banale perché è facile visualizzare una finestra di messaggio che dica "Hello, World". In entrambe le affermazioni troviamo un po' di verità, ma c'è sempre molto da imparare. (Per informazioni sulla proprietà Instancing dei moduli di classe e sul modo in cui Visual Basic tratta classi e collezioni, fate riferimento al Capitolo 14.)
In questo capitolo troverete le informazioni che vi servono per creare applicazioni ActiveX (e comprendere a fondo l'oscuro gergo che circonda la materia).

I diversi tipi di applicazione ActiveX

E perfettamente possibile che un'applicazione sia contemporaneamente un normale eseguibile e un'applicazione ActiveX che espone oggetti utilizzabili da applicazioni client. Per esempio, Excel può essere avviato normalmente e gli oggetti che espone possono essere utilizzati da client OLE. (Gli esempi del capitolo precedente dimostrano come le applicazioni client possono usare Excel come server per calcolare l'ammortamento di un prestito e verificare l'ortografia del testo contenuto in controlli VB.) Le applicazioni che presentano questi due aspetti sono i supereroi in incognito del mondo ActiveX: normali applicazioni "Clark Kent" per la gestione dei

fogli di calcolo di tutti i giorni, e strumenti "Superman" nascosti utilizzabili da altre applicazioni.
Tuttavia, nella realtà di tutti i giorni la maggior parte delle applicazioni scritte in VB6 prenderà una forma o l'altra, raramente entrambe. In fin dei conti, perché preoccuparsi di trasformare Clark Kent in Superman dentro una cabina telefonica se non è realmente necessario? Le applicazioni che da un lato presentano un'interfacia utente completa per l'uso normale e dall'altro espongono oggetti ActiveX per l'uso da parte di client richiedono il doppio del lavoro. La maggior parte delle applicazioni ActiveX scritte in VB è destinata a lavorare nell'anonimato, utilizzata da anplicazioni client senza mai apparire all'utente.

Visto che la maggior parte delle applicazioni ActiveX non prevede un 'interfaccia visibile, il progetto ActiveX predefinito in VB6 include un modulo di classe, ma nessun form. Ovviamente siete liberi di aggiungere a un progetto ActiveX tutti i form che volete.

Visual Basic gestisce automaticamente parte della trafila richiesta per la creazione di oggetti ActiveX: come sempre questo significa dover rinunciare a parte del controllo. Nel contesto del software, "rinunciare al controllo" vuol dire perdere l'accesso a molte delle funzionalità di basso livello.

Server e client, classi e oggetti

Per quanto i termini *server, client, classe* e *oggetto* possano sembrare oscuri, in realtà i concetti che rappresentano sono molto semplici. Una classe è un'impronta, immaginatevela come uno stampino, da cui vengono *create* le istanze della classe. Ogni istanza della classe è un oggetto: istanziare un oggetto significa crearlo basandosi su una classe. In VB6, le classi vengono definite mediante i moduli di classe.
Un server espone (mette a disposizione) uno o più oggetti perché siano usati da altre applicazioni. Le applicazioni che accedono agli oggetti esposti sono client.

È possibile creare tre tipi di applicazioni ActiveX:

- *Server out-of-process,* chiamati anche *server cross-process,* eseguibili VB6 distinti che vengono avviati con un proprio stack in uno spazio di elaborazione distinto.
- *Server in-process,* DLL ActiveX che offrono servizi alle applicazioni client utilizzando lo stack e lo spazio di processo del client. Un server in-process risulta più veloce rispetto ad uno out-of-process analogo, perché non avvengono chiamate inter-processo, ovvero chiamate ad un'applicazione eseguita in un thread distinto. Tuttavia, esistono dei limiti a quello che è possibile inserire in una DLL ActiveX di VB6.
- *Server remoti,* eseguibili VB6 che vengono eseguiti in rete.

La proprietà Instancing dei moduli di classe

Esaminiarno più da vicino la proprietà Instancing dei moduli di classe, che, come abbiamo già detto, contribuisce a stabilire se un'applicazione sia almeno tecnicamente ActiveX: ogni applicazione ActiveX deve includere almeno un modulo di classe per cui la proprietà Instancing sia impostata in modo che la classe sia creabile dall'esterno.
se un modulo di classe non può essere istanziato (perché la proprietà Instancing è stata impostata a 1 - Not Creatable o 2 - PublicNotCreatable), gli oggetti basati sulla classe non potranno essere creati esternamente dalle applicazioni client.

Se la proprietà Instancing *è impostata a 2* - PublicNotCreatable, *è comunque possibile creare gli oggetti internamente al server e successivamente utilizzarli dall'esterno.*

La creazione dall'esterno avviene utilizzando le istruzioni e funzioni Dim...As New, Set e CreateObject. Le applicazioni esterne possono anche manipolare oggetti che non sono in grado di creare direttamente: in questo caso, la stessa applicazione ActiveX deve fornire un metodo (normalmente di nome Add e applicato a una collezione interna) per creare indirettamente gli oggetti. Questi oggetti sono detti *dipendenti,* e verranno trattati nel seguito di questo capitolo.
Se gli oggetti basati su un modulo di classe possono essere creati esternamente (perché la proprietà Instancing del modulo di classe è impostata adeguatamente), le applicazioni client potranno istanziarli. È più comune impostare Instancing a Creatable Multiuse, piuttosto che a Creatable SingleUse. La differenza è il numero di istanze dell'oggetto che il server è in grado di creare: l'impostazione Creatable Multiuse fa in modo che tutte le copie degli oggetti istanziate da una classe vengano create dallo stesso server; Creatable SingleUse, invece, carica in memoria una copia dell'oggetto ActiveX distinta per ogni istanza che viene creata. Evidentemente, la seconda opzione è più costosa in termini di memoria, ma in certe situazioni potrebbe rivelarsi necessaria: visto che le diverse istanze degli oggetti sono controllate da applicazioni ActiveX distinte, un'istanza non può bloccare le chiamate ai metodi di un'altra.

Creazione di un'applicazione ActiveX passo per passo

In questo paragrafo vedremo i passi e alcune delle opzioni del processo di creazione di un'applicazione ActiveX (disponibile sul CD-ROM come ActiveX.Vbp) e di un client, Client. Vbp, che usa le classi da questa esposte.

L'applicazione ActiveX contiene il modulo di classe StringFunctions, che presenta varie proprietà e tre metodi per la manipolazione delle stringhe, che sono funzioni sviluppate in altri capitoli di questo libro. StringFunctions contiene anche un metodo che visualizza un form modale. Le Tabelle 23.2 e 23.3 elencano metodi e proprietà definiti nel modulo di classe StringFunctions.

Tabella 23.2 *I metodi della classe StringFunctions.*

Nome del metodo	Descrizione	Già presentato in questo libro
CapFirstLetter	Mette in maiuscolo la prima lettera di ogni parola in una stringa	nel Capitolo 13
ReverseString	Inverte il contenuto di una stringa	Nuovo
StripText	Sostituisce ogni ricorrenza di un carattere all'interno di una stringa con un altro	nel Capitolo 22

Il modulo di classe StringFunctions è progettato in modo che vengano impostate delle proprietà anziché essere passati dei parametri.

Tabella 23.3 *Le proprietà della classe StringFunctions.*

Proprietà	Descrizione
InReplace	Il carattere che il metodo StripText deve sostituire
InText	La stringa di testo da elaborare
OutText	La stringa di testo al termine dell'elaborazione
OutWith	Il carattere che il metodo StripText deve inserire

Inoltre, le due variabili private InChar e OutChar vengono utilizzate internamente per convertire InReplace e OutWith negli equivalenti codici ASCII. Queste sono le dichiarazioni del modulo di classe:

```
Option Explicit
Public InText As String, OutText As String
Private InChar As Integer, OutChar As Integer
```

CapFirstLetter e ReverseString non ricevono parametri e non restituiscono valori; l'input viene fornito dalla proprietà InText, mentre l'output viene messo nella proprietà OutText, come si vede nel Listato 22.1:

Listato 23.1 *Mettere in maiuscolo la prima lettera e invenire una stringa.*

```
Public Sub CapFirstLetter()
   Dim PosDel As Integer, DeLim As String
   DeLim = " "
   Mid(InText, 1, 1) = UCase(Mid(InText, 1, 1))
   PosDel = InStr(InText, DeLim)
   While PosDel <> 0
      Mid(InText, PosDel + 1, 1)
         UCase(Mid(InText, PosDel + 1, 1))
      PosDel = InStr(PosDel + 1, InText, DeLim)
   Wend
   OutText = InText
End Sub
```

```
Public Sub ReverseString()
Dim intCt As Integer
  Dim strNew As String
  For intCt 1 To Len(Trim(InText))
    strNew = Mid(InText, intCt, 1) & strNew
  next intCt
  OutText = strNew
End Sub
```

Ecco le proprietà della classe che convertono InReplace e OutWith in semplici codici ASCII utilizzabili dal metodo StripText:

```
Public Property Let InReplace(vNewValue As String)
  If vNewValue = "" Then vNewValue = " "
  InChar = Asc(vNewValue)
End Property

Public Property Let OutWith(vNewValue As String)
  If vNewValue = "" Then vNewValue = " "
  OutChar = Asc(vNewValue)
End Property
```

Ed ecco il metodo StripText modificato:

```
Public Sub StripText ()
  Dim StartPos As Integer
  Dim FoundPos As Integer
  StartPos = 1
  FoundPos = InStr(StartPos, InText, Chr(InChar))
  While FoundPos > 0
    Mid(InText, FoundPos, 2) = Chr(OutChar)
    StartPos = FoundPos + 1
    FoundPos = InStr(StartPos, InText, Chr(InChar))
  Wend
  OutText = InText
End Sub
```

Denominazione delle classi ActiveX

E molto importante che i nomi di classi, proprietà, metodi ed eventi siano chiari e facilmente comprensibili per gli utenti dell'applicazione. Microsoft suggerisce le seguenti direttive:

I nomi delle costanti ActiveX devono avere un prefisso basato sul nome del server.

Per i nomi di classi, metodi e proprietà, usare parole intere o l'intera prima sillaba. Per esempio, StringFunctions o Funzioni Stringa e non StrFuncs o FunStr. Le abbreviazioni possono portare a confusione, perché una parola può essere abbreviata in molti modi.

Usare identificatori composti da parole singole con maiuscole/minuscole, come ReverseString.

- Evitare la notazione ungherese.
- Soprattutto, far riferimento agli oggetti esposti e alle loro proprietà e ai loro metodi utilizzando una terminologia comprensibile agli utenti.

Proprietà o parametri?

Da un punto di vista formale, è possibile passare i parametri in una chiamata ad un metodo oppure impostando più proprietà. Immaginate di avere un metodo DoSomethingToName all'interno dell'istanza X di un modulo di classe. Se DoSomethingToName prevede come parametro OldName, lo elabora, e restituisce un valore trasformato il modo canonico per codificare la funzione è:

```
NewName = X.DoSomethingToName(OldName)
```

Se nel modulo di classe avete definito le proprietà OldName e NewName, potete riscrivere DoSomethingToName sotto forma di procedura priva di parametri. A questo punto, DoSomeThingToName leggerebbe il valore della proprietà OldName ed assegnerebbe un valore a NewName. Basterebbe impostare la proprietà OldName, chiamare DoSomeThingToName, quindi usare il valore memorizzato nella proprietà NewName:

```
X.OldName = "I Love Lucy"
X.DoSomethingToName
txtMytext = X.NewName
```

Come esempio pratico, considerate la trasformazione della funzione StripText (dalla sua forma originale, presentata nel Capitolo 22): originariamente, StripText accettava parametri in input; una volta riscritta (nella versione presente nel modulo di classe StringFunctions di questo capitolo) non ha più argomenti, e svolge il suo compito esclusivamente leggendo e impostando delle proprietà.
Lo standard comunemente utilizzato prevede l'utilizzo delle proprietà al posto del passaggio di parametri quando si lavora con oggetti basati su moduli di classe, perché se ne semplifica l'utilizzo. Tuttavia, bisogna tener presente che, nel caso di server out-of-process, ogni impostazione di proprietà porta a uno scadimento delle prestazioni.

La parola riservata Optional *permette di chiamare un metodo con o senza parametri: se non vengono passati i parametri, il metodo imposta le proprietà. (Per maggiori informazioni sull'uso della parola chiave* Optional *si veda il Capitolo 4.)*

Per esempio, potremmo definire degli oggetti basati su un modulo di classe Pizza:

```
Public Topping As String 'Definisce una proprietà

Public Sub Bake (Minutes As Long, Optional Topping As Variant)
' Se è stato passato Topping imposta la proprietà Topping
' utilizzando Me per far riferimento all'oggetto Pizza
    If Not IsMissing(Topping) Then Me.Topping = Topping
```

Prepararsi ad eseguire il server

Per dire a Visual Basic di compilare un programma come componente ActiveX si usa la scheda *Component* della finestra di dialogo *Project Properties,* ed è quello che stiamo per fare con ActiveX.Vbp. Ma prima dobbiamo occuparci di un ultimo aspetto: le applicazioni ActiveX devono essere avviate dalla Sub Main in un modulo di codice. Nulla vieta a un server di visualizzare dei Form (a volte viene anche visualizzato un form all'avvio del server), ma per farlo è necessario creare e distruggere esplicitamente le istanze dei form e non ci si può affidare al caricamento iniziale del form.

Per aggiungere una Sub Main, con il comando *AddModule* del menu *Project* si inserisce nel progetto un normale modulo di codice, quindi si definisce la procedura Sub Main, che può anche essere vuota. Questo è il codice completo per il modulo, compresa la procedura Sub Main, per il progetto ActiveX dimostrativo:

```
Option Explicit
Public Const ProgTitle = _
   "Demo di applicazione ActiveX"

Public Sub Main()
   'Sub Main pro forma per l'applicazione ActiveX
End Sub
```

La scheda *General* della finestra di dialogo *Project Properties* permette di definire la Sub Main come punto i avvio dell'applicazione.

Visualizzare form in un'applicazione ActiveX

I form che vengono visualizzati a partire da oggetti basati su moduli di classe devono essere istanziati utilizzando una variabile, come abbiamo visto nel Capitolo 3 e nel Capitolo 14. Per esempio, il seguente metodo visualizza un'istanza di Form1 all'interno della classe StringFunctions:

```
Public Sub DisplayForm()
   Dim X As New Form1
   X.Show
End Sub
```

Se volete che la vostra applicazione server visualizzi un form all'avvio, potete aggiungere del codice simile alla procedura Sub Main. Notate però che le DLL ActiveX in-process non possono visualizzare form non modali.

Inoltre, fate attenzione ad aggiungere form modali alle applicazioni ActiveX se c'è il rischio che non vengano scaricati: in questo caso, il server risulterà eternamente occupato, bloccando le chiamate degli altri client, e comparirà la finestra di dialogo Component Request Pending.

Impostazione delle opzioni del progetto

Per specificare le impostazioni che trasformano un'applicazione in un componente ActiveX; e definire la descrizione usata dalle altre applicazioni per referenziarla nella finestra di dialogo *References,* si usa la finestra di dialogo *Project Properties.* Le Figure 23.1 e 23.2 mostrano le impostazioni per l'applicazione ActiveX di esempio

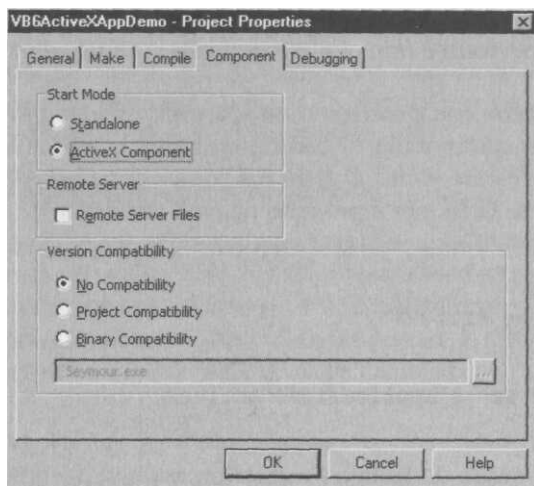
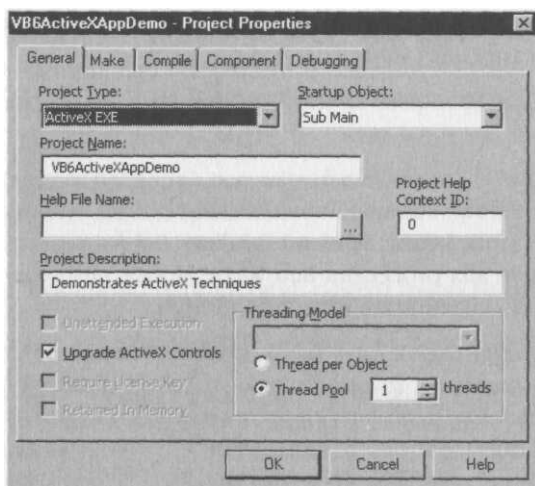
Figura 23.1 *La scheda Component della finestra di dialogo Project Properties permette di indicare al compilatore VB che un progetto è un componente ActiveX.*

L'impostazione di *Start Mode* a *ActiveX Component* invece che a *Standalone* fa in modo che VB mantenga attiva l'applicazione in attesa di una richiesta da parte del client; le applicazioni stand-alone vengono eseguite *e* chiuse immediatamente, a seconda del codice che contengono.

Figura 23.2 *La descrizione delprogetto specificata nella scheda General della finestra Il dialogo Project Properties viene usata per identificare l'applicazione ActiveX nella finestra di dialogo References.*

Usando la scheda *General* della finestra di dialogo *Project Properties* (riprodotta nella Figura 23.2), verificate che l'opzione *Startup Object* sia impostata a *Sub Main* (come abbiamo detto prima). Il nome del progetto definito per l'applicazione, in questo caso *VBóActiveXAppDemo,* comparirà nell'Object Browser,
La descrizione dell'applicazione comparirà anche neH'Object Browser. Inoltre, la descrizione (nel nostro caso "Demonstrates ActiveX Techniques") viene riportata nella finestra di dialogo *References* quando un'altra applicazione vuole utilizzare le classi esposte dal componente ActiveX.

La scheda General *dellafinestra di dialogo* Project *Propertiespermette anche di definire il* modello di threading per *i componenti e i controlli ActiveX. Per un eseguibile ActiveX si può scegliere* Thread per Object, *chefa in modo che ogni istanza di una classe per cui la proprietà* Instancing *è impostata a* Multiuse *venga creata in un thread di esecuzione nuovo e unico, o* Thread Pool, *l'impostazione di default, che fa in modo che ogni classe* Multiuse *venga creata utilizzando a rotazione un thread scelto da unpool. Notate che, in questo caso, ogni threadpossiede un'unica copia di tutte le variabili globali, e più istanze del modulo possono quindi potenzialmente interferire tra loro.*

Se si crea una DLL o un controllo ActiveX, l'elenco *Threading Model* sarà attivato, e permetterà di scegliere tra *single-threaded* e *apartment-threaded.* Il modello Apartment garantisce un certo livello di sicurezza, perché ogni thread si comporta come un appartamento a New York: tutti gli oggetti che crea vivono nell'appartamento e ignorano completamente l'esistenza di oggetti negli altri appartamenti.

Avvio dell'applicazione ActiveX

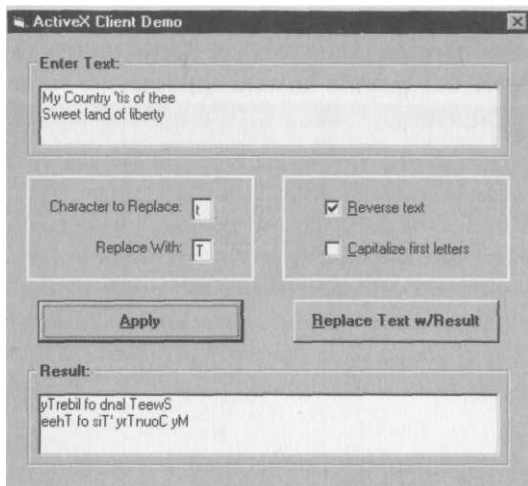
Per collaudare il componente ActiveX, probabilmente vorrete eseguirlo in modalità progettazione e lanciare un'applicazione client in un'altra istanza di Visual Basic.

Assicuratevi di lanciare ilprogetto ActiveX usando il comando Start With Full Compile *(invece di* Start) *del menu* Run. *(Dalla tastiera, questo corrisponde alla combinazione Ctrl+F5, invece che F5.)*

Questa opzione esegue la compilazione dell'intero progetto prima di mandarlo in esecuzione. Se non si facesse così, potrebbero verificarsi errori di compilazione durante la chiamata a oggetti del componente da parte del client, complicando il debugging.

Chiamata del componente ActiveX da un client

La Figura 23.3 mostra il Form utilizzato da Client.Vbp.

Figura 23.3 *// client diprova chiama metodi eproprietà esposti daun'istanza della classe StrìngFunctions.*

Il pulsante *Apply* chiama tutti i metodi e le proprietà di un'istanza della classe, tranne DisplayForm: il Listato 23.2 mostra come.

Listato 23.2 *Chiamate alleproprietà ed ai metodi di una classe ActiveX.*

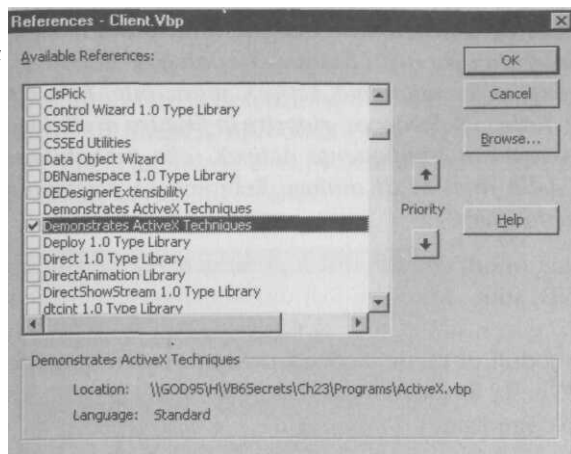
```
Private Sub cmdApply_Click()
   Dim x As New StringFunctions
   x.InText = txtManipulate
   x.InReplace = txtReplace
   x.OutWith = txtWith
   x.StripText
   If chkCap Then x.CapFirstLetter
   If chkReverse Then x.ReverseString
   lblResult = x.OutText
   Set x = Nothing
End Sub
```

Impostazione dei riferimenti nel progetto client

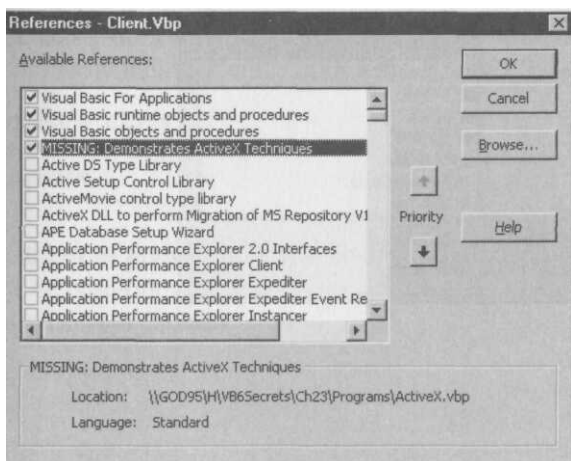
Perché il riferimento alla classe OLE

Dim X as New StringFunctions

possa funzionare, l'applicazione ActiveX deve essere disponibile (e selezionata) nella finestra di dialogo *References* (accessibile dal menu *Project),* come si vede nella Figura 23.4.

Figura 23.4
La finestra di dialogo References permette di collegare un client a un'applicazione ActiveX

Come scoprirete lavorando con server nella modalità di progettazione di Visual Basic, la connessione tra server e client impostata dalla finestra di dialogo References *può risultare molto precaria: potreste dover fermare e riavviare il server un ceno numero di volte prima che il riferimento compaia correttamente nella finestra di dialogo* References *del client.*

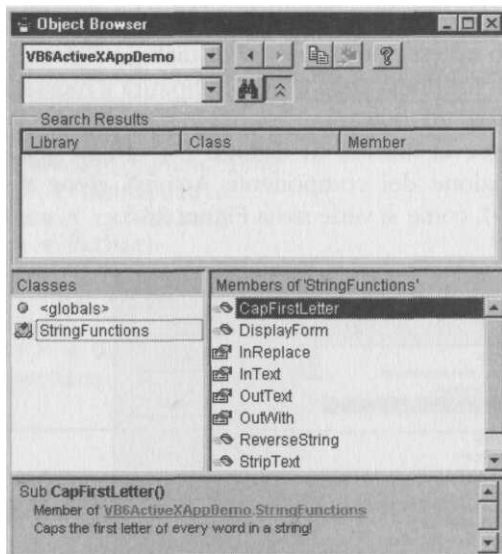
Per esempio, se l'applicazione ActiveX non è in esecuzione, o se la modificate, il client non sarà più connesso ad essa nella finestra di dialogo *References,* e se cercherete di chiamare metodi o proprietà della classe comparirà il messaggio di errore "User-defined type not defined" ad indicarlo.
In questi casi, quando si apre la finestra di dialogo *References* dell'applicazione client, a fianco della descrizione del componente ActiveX viene visualizzata la parola "MISSINO" (mancante), come si vede nella Figura 23.5.

Figura 23.5
Se un 'applicazione ActiveX non è più in esecuzione in un'istanza di VB, o è stata modificata, il riferimento potrebbe non essere più disponibile.

Perché il client possa tornare a funzionare bisognerà riavviare il componente ActiveX (se necessario), deselezionare nella finestra di dialogo References *dell'applicazione client il riferimento al componente ActiveX mancante, fare clic su* OK per *uscire dalla finestra di dialogo* References, *riaprire la finestra di dialogo* Refererreces *trovare il nuovo riferimento al componente ActiveX (che probabilmente compare nell'elenco alfabetico della finestra di dialogo* ReferencesJ, *selezionarlo, premere* OK, *e riavviare il progetto client.*

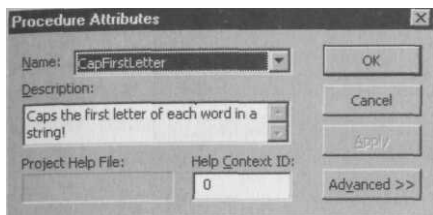
I riferimenti, e i vantaggi offerti dal sistema di gestione delle versioni previsto per i componenti ActiveX VB, sono argomento di uno dei prossimi paragrafi di questo capitolo.
Gli oggetti basati sui moduli di classe ActiveX possono essere creati esternamente anche senza includerli nella finestra di dialogo *References,* a patto che il componente ActiveX sia stato compilato.

Uso dell'Object Browser

Una volta che avete aggiunto al progetto client il riferimento alla libreria di oggetti ActiveX usando la finestra di dialogo *References,* l'Object Browser vi permette di visualizzare i membri delle classi, come si vede nella Figura 23.6.

Figura 23.6
L'Object Browser del progetto client permette di visualizzare metodi e proprietà dei moduli di classe referenziati.

Uso della finestra di dialogo *Procedure Attributes*

La finestra di dialogo *Procedure Attributes* dell'applicazione del componente ActiveX permette di impostare la descrizione dei membri della classe (riportata in fondo alla Figura 23.6). La finestra, riprodotta nella Figura 23-7, è accessibile dal menu *Tools*. Verificate che il membro per cui volete definire descrizione e attributi sia sele- selezionato nella casella di riepilogo *Name.*

Figura 23.7 *Lafinestra di dialogo procedure Attributes permette di impostare gli attributi dei membri.*

Se si apre lafinestra di dialogo Procedure Attributes *mentre il cursore si trova in una procedura all'interno del Code Editor, per default questa sarà selezionata procedure nell'elenco* Name. *Se il Code Editor non è aperto, il comando* Procedure Attributes *è disabilitato.*

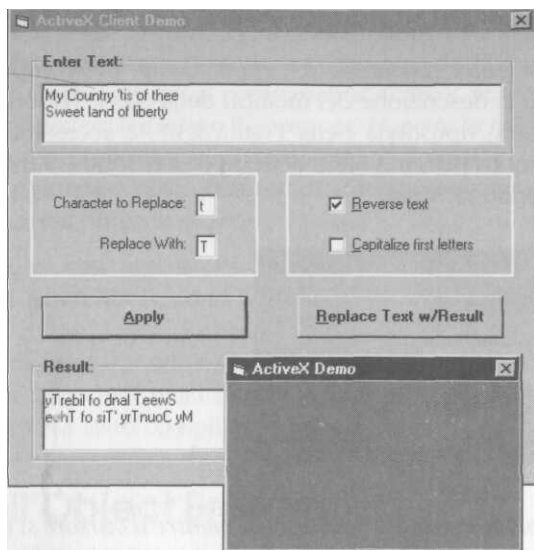
La finestra di dialogo *Procedure Attributes* permette di impostare la descrizione per membri specifici elencati nell'Object Browser. Per esempio, il testo "Caps the first letter of every word in a string!" definito nella casella *Description* in Figura 23.6 compare come descrizione del metodo in Figura 23.7. Inoltre, questa finestra permette di impostare gli Help Context ID (identificatori del contesto della Guida).

Visualizzazione di un form: il client

Come si vede nella Figura 23.8, usare un metodo di classe di un'applicazione ActiveX per visualizzare un form del server da un client è facile: il metodo DisplayForm viene chiamato esattamente come gli altri metodi della classe:

```
Private Sub Form_Click()
    Dim x As New StringFunctions
    x.DisplayForm
End Sub
```

I form visualizzati in questo modo possono essere utilizzati praticamente ad ogni scopo,anche se potrebbe essere necessario realizzare opportuni metodi e proprietà Per consentire l'accesso ad essi.

Figura 23.8
Iprogetto client ha chiamato il metodo ShowForm del server per visualizzare un form.

Creazione di un oggetto senza usare la finestra di dialogo References

È possibile creare oggetti senza collegarli al progetto utilizzando la finestra di dialogo *References* (il modo in cui in generale abbiamo effettuato la connessione a componenti ActiveX negli esempi del Capitolo 22). La sintassi che si usa dimensiona una variabile come l'oggetto desiderato, quindi chiama la funzione `CreateObject`. Perché sia possibile collegarsi agli oggetti di un'applicazione ActiveX's utilizzando `CreateObject`, l'applicazione deve essere compilata: in caso contrario si cercherebbe di connettersi a un'istanza dell'IDE di VB. Quando l'applicazione ActiveX viene eseguita in modalità di progettazione, VB si occupa della registrazione degli oggetti OLE del componente; quando l'applicazione ActiveX viene compilata indipendentemente, i suoi oggetti vengono automaticamente registrati.

In questo ambito, è indifferente compilare l'applicazione in pseudocodice oppure in codice nativo.

Notate che se distribuite i componenti ActiveX la vostra routine di installazione dovrà gestirne la registrazione automatica sui sistemi di destinazione; si veda il Capitolo 35, e il riquadro nel seguito di questo capitolo.

Quando si usa la funzione CreateOb j ect, *per far riferimento all'applicazione ActiveX si utilizza il nome del progetto definito nella scheda* General *della finestra di dialogo* Project Properties, *e non, come ci si potrebbe aspettare, il nome del file eseguibile Per esempio, se avessimo compilato il programma ActiveX di esempio come Seymour.Exe, gli oggetti creati in base al modulo di classe* StringFunctions *verrebbero cornunque referenziati come* VBGActiveXAppDemo.StringFunctions, *perché al progetto è stato attribuito il nome interno* VB6ActiveXAppDemo *usando la finestra di dialogo* Project Properties *(a volte si indica questo identificatore come ProglD).*

Glii oggetti OLE appartenenti al programma Seymour.Exe vengono inseriti nel Registro all'atto della compilazione. Per referenziare gli oggetti del server, usate la funzione CreateObject e assicuratevi di aver disattivato il riferimento al server in modalità progetto dalla finestra di dialogo *References* dell'applicazione client. Infine, modificate le prime righe dell'evento cmdApply_Click:

```
Private Sub cmdApply_Click()
    Dim x As Object
    Set x = CreateObject("VBSActiveXApp.StringFunctions")
    x.InText = txtManipulate
    x.InReplace = txtReplace
    x.OutWith = txtWith
    x.StripText
    If chkCap Then x.CapFirstLetter
    If chkReverse Then x.ReverseString
    lblResult = x.OutText
    Set x = Nothing
End Sub
```

A proposito, nulla vieta di usare la finestra di dialogo *References* per includere un'applicazione ActiveX compilata che avete creato in un'applicazione client. In questo caso userete la ben nota sintassi

```
Dim X as New StringFunctions
```

vista prima. Per informazioni sulle conseguenze dei diversi modi di collegarsi a un server si veda il paragrafo "Binding" che segue.

La funzione GetObject

La funzione GetObj ect, sorella di CreateObject, può essere utilizzata quando è già attiva un'istanza dell'oggetto, o se si vuole creare un oggetto in cui sia già caricato un file. La sintassi usata per referenziare gli oggetti collegati con GetObject è la stessa usata per quelli creati da CreateObject. Tuttavia, non è possibile usare GetObject per referenziare un oggetto basato su una classe creata lanciando un progetto nell'IDE di Visual Basic.

Aggiunta e rimozione della registrazione dei componenti ActiveX

Le applicazioni ActiveX possono essere registrate in tre modi

- Compilando l'applicazione in un file eseguibile
- Eseguendo l'applicazione compilata la prima volta
- Eseguendo l'applicazione compilata con il parametro /Regserver sulla riga di comando, per esempio:

```
C:\VB\Seymour.Exe /Regserver
```

Se si lancia un'applicazione ActiveX con l'opzione /Regserver, l'esecuzione termina subito dopo il completamento della registrazione. Senza il flag, l'applicazione resterebbe attiva.
Inoltre, come già detto, i componenti ActiveX dovrebbero essere registrati automaticamente quando vengono installati sulle macchine target. I programmi di installazione come il Package and Deployment Wizard della Microsoft generalmente si occupano anche di questo per voi. Per maggiori informazioni, si veda il Capitolo 35.
La rimozione della registrazione dei componenti ActiveX è altrettanto importante: se ci si limitasse a cancellare il file eseguibile dal disco rigido, comunque resterebbero le definizioni nel Registro. Per eliminare anche queste, prima di cancellare l'eseguibile dell'applicazione OLE, lanciatelo con il parametro /UnRegserver, per esempio:

```
C:\VB\Seymour.Exe /UnRegserver
```

Non c'è nessun buon motivo per lasciare che questi piccoli noiosi server OLE di prova vi riempiano il Registro, se non vi servono più!

Binding

Abbiamo appena detto che gli oggetti basati sulle classi ActiveX di Visual Basic possono essere creati in due modi. Se si include nel progetto client un riferimento al server utilizzando la finestra di dialogo *References, è* possibile dichiarare una variabile come nuova istanza di una delle classi del server:

```
Dim X As New StringFunctions
```

In alternativa, si può dichiarare la variabile come oggetto e usare la funzione CreateObject e l'istruzione Set per caricare il riferimento ad un oggetto specifico basato su una classe:

```
Dim X As Object
Set X = CreateObject("VBSActiveXAppDemo.StringFunctions")
```

Qual è la fondamentale differenza tra questi due metodi di connessione al client di un'istanza di una classe ActiveX? Il tutto si riduce a una questione di binding (associazione). Il concetto di binding sarà familiare a quelli che hanno già lavorato in ambienti completamente orientati agli oggetti: sostanzialmente riguarda il momento

in cui gli eventi, le proprietà e i metodi vengono assegnati ad un oggetto. In VB6 questo concetto viene applicato ai riferimenti ad oggetti ActiveX: in che momento viene verificata la validità delle chiamate a metodi e proprietà fatte nel codice?
Se si dichiara una variabile come oggetto e successivamente la si associa ad un oggetto mediante la funzione CreateObject, VB applica il cosiddetto *late binding (*associazione tardiva): questo significa che il compilatore non sarà in grado di stabilire se le chiamate a metodi e proprietà siano valide finché il codice non verrà eseguito. Per esempio, il codice

```
Dim x As Object
Set x = CreateObject("VB6ActiveXAppDemo.StringFunctions")
X.ThisMethodDoesntReallyExistHaHa
```

viene compilato senza problemi, ma ovviamente genererà un errore di runtime. Per effetuare l'associazione tardiva, VB deve includere nell'eseguibile del codice per verificare la validità delle chiamate a metodi e proprietà, ottenere l'identificatore ActiveX necessario per chiamare il metodo e generare gli errori. L'intero processo prevede la ricerca delle funzioni in una tabella virtuale per recuperarne l'identificatore (La tabella virtuale di un oggetto e delle sue funzioni è anche detta *vtable).* Se Visual Basic non riesce a stabilire esattamente il tipo di oggetto che si sta utilizzando, non può determinare quale *vtable* usare.
Il late binding, quindi, è il modo più costoso (in termini di tempo e risorse utilizzate) per referenziare gli oggetti delle classi ActiveX. Eppure, in certi casi, può tornare utile: per esempio, potreste non sapere che tipo di oggetto verrà referenziato da una variabile fino all'esecuzione del programma.
L'early binding (associazione precoce) viene applicato quando VB riesce a riconoscere in fase di compilazione a quale oggetto appartengano i metodi e le proprietà. In questa situazione, il codice compilato può contenere solo la chiamata del metodo, e il codice necessario per la ricerca nella *vtable* e il controllo degli errori può essere eliminato. Ne risulta un notevole incremento delle prestazioni rispetto al late binding.
Per utilizzare l'associazione precoce è necessario aggiungere nel progetto client un riferimento all'applicazione ActiveX selezionando il server nella finestra di dialogo *References.* Una variabile dichiarata come nuova istanza di una classe referenziata del server, come per esempio

```
Dim X As New StringFunctions
```

può contenere solo un oggetto della classe StringFunctions, e VB sarà in grado di determinare gli identificatori delle funzioni ActiveX (o se il metodo o la proprietà esistano del tutto) in fase di compilazione. Il codice

```
Dim X As New StringFunctions
  ThisMethodDoesntReallyExistHaHa
```

genera un errore di compilazione. Ride bene chi ride ultimo!
Notate che l'early binding si basa sull'identificatore univoco della classe (CLSID) del server, che è un identificatore esadecimale definito quando il server viene compilato. Il nome della classe (o del progetto) non viene utilizzato perché potrebbero

esistere più classi con lo stesso nome ma con metodi e proprietà differenti. Per avere un'idea di come si presenti un CLSID potete dare un'occhiata al Registro con Regedit. Per maggiori informazioni su CSLID e binding fate riferimento ai Capitoli 9 e 14.

Codice per gli eventi di una classe

I moduli di classe prevedono due eventi:

- Initialize, che si verifica quando viene creata un'istanza della classe
- Terminate, che si verifica quando l'istanza viene distrutta

Questi eventi sono esaminati più in dettaglio nel Capitolo 14. Potete facilmente aggiungere del codice per visualizzare un messaggio quando si verificano questi eventi, per esempio:

```
Private Sub Class_Initialize()
   MsgBox "Eeek! Performance anxiety: I've Been Initialized!",
      vbExclamation, ProgTitle
End Sub

Private Sub Class_Terminate()
   MsgBox "Terminating is hard to do, yes it is!",
      vbExclamation, ProgTitle
End Sub
```

La finestra di messaggio *Initialize* viene visualizzata all'atto della creazione di un oggetto basato sulla classe StringFunction in seguito alla pressione del pulsante *Apply,* come si vede nella Figura 23-9.

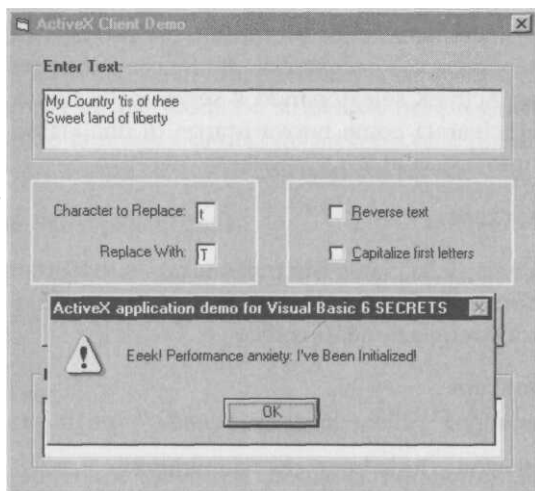
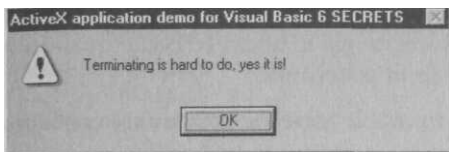
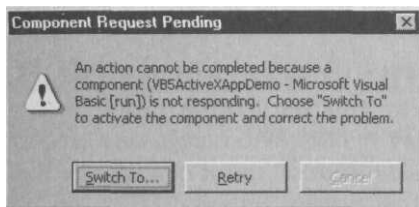
Figura 23.9 *Le applicazioni client lanciano codice definito nell'evento Initialize del modulo di classe quando creano oggetti basati su di esso.*

Quando si distrugge l'istanza di una classe, nel nostro caso uscendo dal contesto della funzione che l'ha creata, viene visualizzata la finestra di messaggio *Terminate, come* si vede nella Figura 23.10.

Figura 23.10
La finestra collegata all'evento.

La finestra di dialogo Component Request Pending

Se provate a chiamare un oggetto istanziato dalla classe `StringFunctions` prima di premere OK nelle finestra di messaggi riprodotte nelle Figure 23.9 e 23.10, probabilmente otterrete l'odiata finestra di dialogo *Component Request Pending,* riprodotta nella Figura 23.11.

Figura 23.11
Se si cerca di accedere ad un server occupato, compare la finestra di dialogo Component Request Pending.

La finestra di dialogo Component Request Pending *era già diventata famosa come* Server Busy. *Se avete mai passato un po' di tempo a lavorare con componenti ActiveX come server, vi sarete accorti che durante il processo di sviluppo tendono a sovraccaricarsi, e generare la finestra* Component Request Pending.

Poiché le finestre di messaggio generate dalle funzioni `MsgBox` inserite negli eventi della classe ActiveX sono modali, il server va in stallo, e attende che l'utente risponda al messaggio prima di continuare. In questa situazione, i pulsanti presenti nella Figura 23.11, *Switch To* e *Retry,* non servono a granché. *Retry* continuerà a provare all'infinito senza alcun esito finché qualcuno non premerà *OK* sulla finestra di messaggio modale (che nel frattempo sarà anche scomparsa), e la finestra *Component Request Pending* continuerà a riapparire. *Switch To* visualizza la finestra del Task Manager di Windows. Sfortunatamente, il componente ActiveX, non avendo un'interfaccia visibile, non sarà nell'elenco delle applicazioni a cui è possibile passare o che si possono chiudere.

La finestra di dialogo *Component Request Pending* viene generata tanto spesso dai server che, se non ha ancora incrociato la vostra strada nei vostri contatti coi componenti ActiveX, potete star certi che lo farà presto! Fortunatamente, è possibile impostare alcune opzioni per l'applicazione client in modo che la finestra sia un po' più amichevole.

Per cominciare, non ha senso includere il pulsante *Switch To* se già sappiamo che non servirà a nulla, a parte confondere gli utenti. Notate che, nella Figura 23.11 il pulsante *Cancel* è disattivato. Tecnicamente, il pulsante *Cancel* è attivato quando la finestra *Component Request Pending* viene visualizzata perché il server è occupato (è una situazione analoga al segnale di occupato al telefono), e disattivato quando viene visualizzata perché una richiesta è già in attesa (è come quando vi lasciano in attesa con quelle orribili musichette in sottofondo).

La questione è che una richiesta in attesa potrebbe comunque essere trattata, se chi chiama ha la pazienza di aspettare nonostante la pessima musica. Se invece la chiamata restituisce "server busy" non c'è niente da fare: bisognerà richiamare.

Da un punto di vista funzionale, la finestra di dialogo *Component Request Pending* compare in risposta ad una situazione di attesa della richiesta con il pulsante *Cancel* disattivato, e su di essa rimane un solo pulsante attivo, che però non fa assolutamente niente. In pratica la finestra si riduce ad essere una specie di finestra di messaggio con un pulsante di OK (o, come fanno notare i più pignoli, un pulsante di "Non OK"): perché allora non farla apparire per quello che è?
L'oggetto App dell'applicazione client espone una serie di proprietà utilizzabili per personalizzare l'aspetto della finestra di dialogo *Component Request Pending.* Per esempio, se si inserisce nell'evento Load del form del client il seguente codice,

```
Private Sub Form_Load()
   App.OleRequestPendingMsgText =
      "Hacker's Component Request Pending Dialog"
   App.OleRequestPendingMsgTitle = "Not OK"
End Sub
```

la finestra *Component Request Pending* che ne risulta è quella riprodotta nella Figura 23.12.

Figura 23.12 *Personalizzazion con le proprietà dell 'oggetto App.*

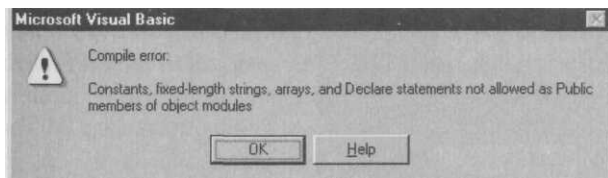
Nella vita reale, probabilmente il messaggio potrebbe suggerire all'utente di contattare il servizio di supporto tecnico, o qualcos'altro.
È anche possibile modificare l'aspetto della finestra *Component Request Pending* in caso di server occupato: in questo caso, sulla finestra saranno disponibili i pulsanti *Cancel* e *OK.* Il testo e il titolo sono controllati dalle proprietà OleServerBusyMsgText e OleServerBusyMsgTitle dell'oggetto App.
Le proprietà OleServerBusyTimeOut e OleRequestPendingTimeOut, impostabili in esecuzione, permettono invece di modificare i valori predefiniti per il tempo di attesa prima che venga visualizzata la finestra *Component Request Pending.* L'accorgimento può tornare utile se prevedete che il server resti occupato per periodi considerevoli, per esempio perché deve caricare database di grandi dimensioni: si

potrebbe evitare la comparsa della finestra *Component Request Pending* almeno nei casi in cui non è necessaria.
Nel caso del server occupato, è possibile evitare del tutto la finestra *Component Request Pending* impostando a True la proprietà OLEServerBusyRaiseError dell'oggetto App e controllando l'errore OLE Server busy (vedi il codice seguente): il tutto equivale alla pressione del pulsante *Cancel* della finestra *Component Request Pending* da parte dell'utente.

```
App.OleServerBusyRaiseError = True

On Error GoTo ErrHandle
Const OleServerBusyError = &H80010001

Exit Sub
ErrHandle:
   If Err = OleServerBusyError Then
      MsgBox "Quitting...", vbCritical, "OLE Error"
   Else
      'Gestisce altri errori
   End If
   'Pulisce il codice secondo necessità
End Sub
```

Un modulo di classe è un involucro

È facile aggiungere un modulo di classe che funga da involucro (o interfaccia) per normali moduli di codice o form. Potreste volerlo fare semplicemente per trasformare in server librerie di codice già sviluppate. Esistono però alcuni vincoli a quello che si può mettere nei moduli di classe: se si cercano di inserire in un modulo di classe variabili o dichiarazioni di funzioni globali, viene generato un errore di compilazione analogo a quello documentato dalla Figura 23.13.

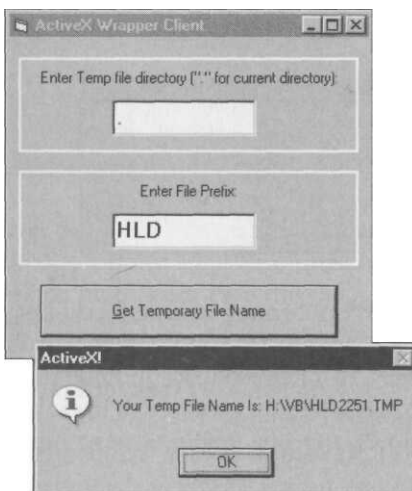
Figura 23.13
Se volete usare costanti o dichiarazioni esterne globali, dovrete inserirle al di fuori del modulo di classe.

Come dimostrazione, il server Wrapper.Vbp fornisce un'interfaccia per il modulo di codice File_Utilities sviluppato nel Capitolo 16. Sono state incapsulate due delle funzioni di File_Utilities: GetTempFile, che restituisce il nome di un file temporaneo, e FixFile, che, se necessario, aggiunge un carattere di backslash in fondo al percorso del file. (Si veda il Capitolo 22 per una spiegazione completa della funzione GetTempFile.)

Per usare l'involucro da un client, è necessario chiamare il progetto Wrapper e avviarlo come componente ActiveX, con *Start With Full Compile,* dalla Sub Main come abbiamo detto precedentemente. Il modulo di classe si chiama Files, e la sua proprietà Instancing è impostata a 5-Multiuse. Ecco la funzione involucro che chiama il modulo di codice FILE_UTILITIES:

```
Public Function GetTempFile(Directory As String, _
   Prefix As String) As String
   GetTempFile = FILE_UTILITIES.GetTempFile _
      (FILE_UTILITIES.FixPath(Directory),Prefix)
End Function
```

Per chiamare il componente ActiveX, verificate che Wrapper sia selezionato nella finestra di dialogo *References* (il client è disponibile come WrapCli.Vbp). Troverete il componente ActiveX nell'elenco della finestra di dialogo *References* descritto come "Wraps File Utility Module". Chiamare il componente Wrapper è facile, come dimostra la Figura 23.14:

```
Private Sub cmdGet_Click()
   Dim FileName As String
   Dim X As New Files
   FileName = X.GetTempFile(txtDirect, txtPrefix)
   Set X = Nothing
   If FileName = "" Then
      MsgBox "Unable to create a temp file!", vbCritical, _
         "ActiveX!"
   Else
      MsgBox "Your Temp File Name Is: " & FileName, _
         vbInformation, "ActiveX!"
   End If
End Sub
```

Figura 23.14
I moduli di classe ActiveX possono essere utilizzati per incapsulare moduli di codice esistenti.

Gestione degli errori con componenti ActiveX

Errorinon intercettati in un componente ActiveX, o errori generati di proposito con il metodo Raise dell'oggetto Err, vengono riprodotti nell'applicazione client che ha chiamato il componente. (Per informazioni più generali sull'oggetto Err e il metodo Raise, consultate il Capitolo 15).
Di conseguenza, nel caso dei componenti ActiveX non basta gestire l'errore visualizzando una finestra di messaggio: il server deve restituire al client le informazioni per la gestione degli errori, in modo che il client possa decidere se, e cosa, fare.
Per esempio, il modulo di classe Wrapper descritto nel paragrafo precedente dovrebbe almeno verificare che alla funzione GetTempFile venga passato un prefisso di file valido. In mancanza di un prefisso valido, la chiamata API fallirà, l'ActiveX anche, e al client arriverà un oscuro messaggio, o addirittura nessun messaggio, se il thread va in stallo. Si potrebbe aggiungere del codice che genera un errore in mancanza di un prefisso valido:

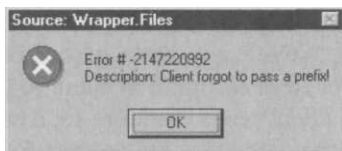
```
Const hdNoPrefixError = 512
If Prefix = "" Then
   Err.Raise Number:=vbObjectError + hdNoPrefixError, _
   Description:="Il client ha dimenticato il prefisso!", _
   Source:="Wrapper.Files"
End If
```

(Ovviamente, in realtà non dovremmo limitarci a controllare l'esistenza della stringa: il prefisso dovrebbe essere composto da tre caratteri validi, e così via.) vbObjectError è una costante predefinita (il cui valore è -2,147,221,504). Le costanti che create per indicare gli errori dei vostri server dovrebbero partire dal valore vbObjectError + 512. (Possono arrivare fino al valore vbObjectError + 65535). È importante documentare in maniera esaustiva i codici di errore per gli utenti degli oggetti del vostro componente ActiveX. Quando generate un errore, dovreste anche fornire una descrizione del problema nella proprietà Description dell'oggetto Err ed includere le informazioni sulla fonte nella proprietà Source sotto forma di ProgID (cioè oggetto.classe). L'applicazione client può quindi intercettare specifici errori e trattarli. Per esempio:

```
On Error GoTo ErrHandle

Exit Sub
ErrHandle:
   With Err
   If .Number = vbObjectError + hdNoPrefixError Then
      MsgBox "Errore # " + CStr(.Number) + vbCrLf + _
         "Descrizione: " + .Description + vbCrLf, vbCritical, _
         "Sorgente: " + .Source
      End If
```

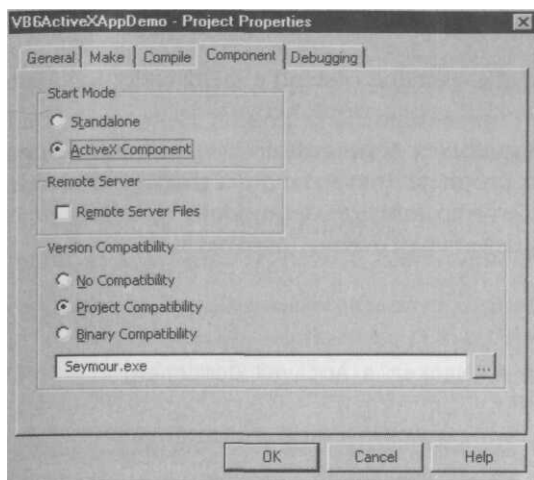
```
    End With
End Sub
```

Evidentemente, in questo caso non facciamo altro che visualizzare la finestra di messaggiriprodotta in Figura 23.15. Potremmo gestire l'errore riprovando con un prefisso valido, o impostando il focus sulla casella del prefisso.

Figura 23.15
Un errore in un componente ActiveX intercettato dall'applicazione client che lo ha chiamato.

Se provate ad eseguire i programmi involucro in versione non compilata, l'errore viene intercettato dai gestori di default di VB. In questo caso, anche se VB mostrerà un codice e una descrizione per l'errore, la finestra visualizzata non corrisponderà a quella riprodotta nella Figura 23.15.

Gestione delle versioni di un componente ActiveX

I componenti ActiveX normalmente vengono distribuiti inseme ad altre applicazioni compilate. Questo potenzialmente può portare a problemi se modificate i vostri componenti ActiveX. Le applicazioni che li utilizzano continueranno a funzionare anche con la nuova versione? Ci si augura sempre che l'aggiornamento del server sia richiesto dagli sviluppatori che lo utilizzano e ne sono così entusiasti da acclamare l'aggiunta di nuove e migliori funzioni.
Fortunatamente, VB fornisce una funzionalità di supporto per la gestione delle versioni dei componenti ActiveX. Il trucco consiste nell'inserire il percorso completo della versione precedente dell'ActiveX nel campo *Compatible ActiveX* della scheda *Component* della finestra di dialogo *Project Properties.* VB tiene traccia di classi e membri del componente ActiveX utilizzando un numero interno per la libreria di tipi. In Figura 23.16, Seymour.Exe è la versione compilata di ActiveX.Vbp sviluppata in questo capitolo. Una volta che avete compilato la versione 1.0 del progetto — come Seymour.Exe — dovreste mantenerlo come componente ActiveX compatibile finché non avrete a disposizione la versione di aggiornamento completa del server, per evitare di instasare il sistema di controllo delle versioni con troppi build intermedi.

Figura 23.16 *Le imposizioni per Verston Campatibility nella scheda Component della finestra di dialogo Project Properties permettono il monitoraggio delle versioni dei componenti ActiveX.*

Creazione di un oggetto applicativo

Le applicazioni ActiveX commerciali che abbiamo utilizzato nel Capitolo 22 offrono tutte un oggetto Application al livello più alto della gerarchia. Tipicamente, gli oggetti Application contengono informazioni sull'applicazione nel suo complesso e permettono di accedere ai livelli inferiori della gerarchia di oggetti di un'applicazione. Includere un oggetto Application in un ActiveX non è strettamente necessario, ma se decidete di farlo, dovreste almeno prevedere le seguenti proprietà:

- Se l'ActiveX ha un'interfaccia utente, una proprietà Caption per impostare o recuperare la didascalia della finestra principale dell'applicazione server
- Una proprietà Name a sola lettura
- Una proprietà Path che restituisca il percorso dell'eseguibile dell'applicazione
- Una proprietà Version che restituisca proprietà le informazioni sulla versione che VB compila insieme all'eseguibile

Una cosa che bisognerebbe evitare di implementare in un oggetto Application è un metodo per chiudere il server (Sì, sì, Rex, lo so che Excel ha un metodo Quit, Rex. Buono, Rex. Rex, facciamo quello che la Microsoft dice, non quello che fa!). Questo perché, almeno teoricamente, un ActiveX "educato" non controlla la propria esistenza: solo il client può farlo. Il server dovrebbe restare attivo finché l'applicazine client conserva un riferimento valido ad un suo oggetto. Se si chiude il server internamente si rischia di lasciare il client con riferimenti ad oggetti che non esistono più. Ho creato un ActiveX di prova (App.Vbp sul CD-ROM) con un oggetto Application che implementa le quattro proprietà appena viste come proprietà della classe. L'applicazione è impostata per girare come componente ActiveX con il nome Hacker. Il progetto contiene:

- Un form, frmMain, che rappresenta il form principale di un'applicazione ActiveX
- Un modulo di codice che contiene costanti e dichiarazioni globali e la Sub Main fittizia per l'avvio dell'applicazione ActiveX
- Un modulo di classe pubblico, Application, che definisce l'oggetto Hacker.Application. La proprietà Instancing del modulo di classe è impostata a 5-Multiuse. L'evento Initialize del modulo visualizza un'istanza di frmMain, che viene scaricata nell'evento Terminate.

Ecco il modulo di codice:

```
Option Explicit
Public Const hdAppName = "Hacker's ActiveX Component"
Public X As New frmMain

Public Sub Main()
   'Sub Main pro forma per l'avvio del componente ActiveX!
End Sub
```

Come si vede nel Listato 23-3, il modulo di classe Application contiene il codice delle proprietà Caption, Name, Path e Version:

Listato 23.3 *Creazione di un oggetto ActiveX Application.*

```
Option Explicit

Public Property Get Caption() As String
   Caption = X.Caption
End Property

Public Property Let Caption(vNewValue As String)
   X.Caption = vNewValue
End Property

Property Get Name() As String
   Name = hdAppName
End Property

Property Get Path() As String
   Path = App.Path
End Property

Property Get Version() As String
   Version = App.Major & "." & App.Minor
End Property
```

X, l'istanza di frmMain utilizzata dal server, viene visualizzata nell'evento Initialize del modulo di classe, e scaricata nell'evento Terminate:

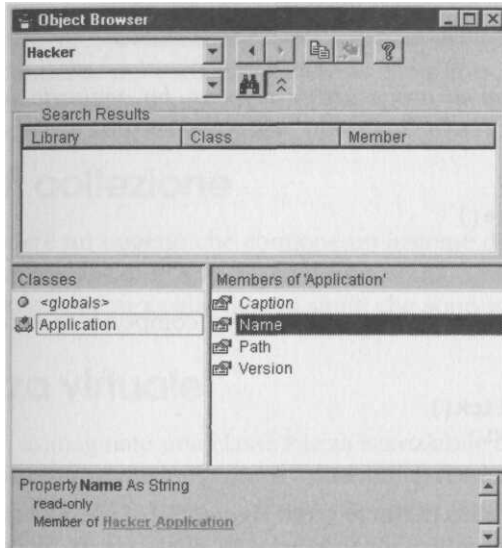
```
Private Sub Class_Initialize()
   X.Show
End Sub

Private Sub Class_Terminate()
```

```
    Unload X
End Sub
```

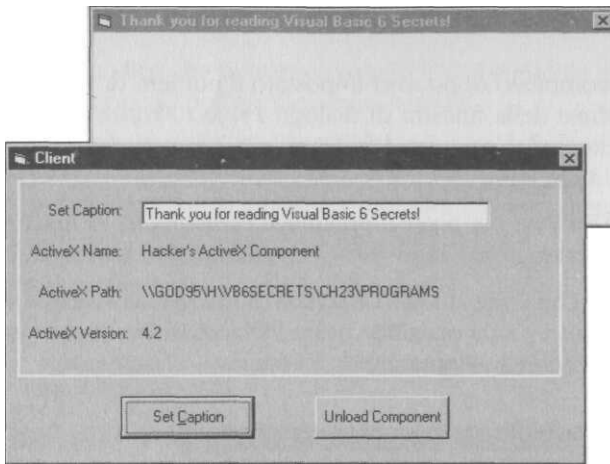
Il progetto è stato compilato dopo aver impostato il numero di versione a 4.2 utilizzando la scheda *Make* della finestra di dialogo *Project Properties.* Il passo succes- è creare un client che possa comunicare con l'oggetto `Application`: si tratta del progetto client AppClient.Vbp.

Ovviamente, come sempre, per poter utilizzare il componente ActiveXApplication sul vostro sistema dovrete prima registrarlo, compilandolo o lanciandolo.

Dopo aver verificato che l'oggetto `Application` di Hacker sia selezionato nella finestra di dialogo *References,* sarà possibile usare l'Object Browser per visualizzarne le proprietà, come si vede nella Figura 23.17.

Figura 23.17 *Se l'oggetto Application di Hacker rientra nell'elenco dei riferimenti disponibili, si può usare l'Object Browser per visualizzare le proprietà di Application.*

Il client di prova che ho scritto per l'applicazione utilizza il late binding per creare l'oggetto server, quindi se volete potete rimuovere l'oggetto `Application` di Hacker dall'elenco dei riferimenti attivi. Il client, visibile in Figura 23.18, prevede un unico form che crea una variabile a livello di form per contenere l'istanza dell'oggetto del componente ActiveX. Ecco la dichiarazione a livello di form e il codice per l'evento `Load`:

```
Option Explicit
Dim X As Object

Private Sub Form_Load()
   Set X = CreateObject("Hacker.Application")
   'riempire il form sulla base dei valori del server
   lblName = X.Name
   lblPath = X.Path
   lblVersion = X.Version
   txtCaption = X.Caption
End Sub
```

Figura 23.18
Un'applicazione client può facilmente leggere ed impostare le proprietà dell'oggetto Hacker. Application.

Per essere sicuro che il form di prova avesse il focus, ho aggiunto un comando Me.SetFocus *nell'evento* Activate *(se l'avessi messo nell'evento* Load *si sarebbe verificato un errore):*

```
Private Sub Form_Activate()
   Me.SetFocus
End Sub
```

Il codice che imposta la caption del form del nuovo componente ActiveX è veramente banale:

```
Private Sub cmdSetCap_Click()
   X.Caption = txtCaption
End Sub
```

Sul form di prova trovate un altro pulsante (vedi Figura 23.18) che alterna tra *Unload Component* e *Load Component,* a seconda che X contenga o meno un'istanza dell'oggetto Hacker.Application:

```
Private Sub cmdUn_Click()
   If Not X Is Nothing Then
      Set X = Nothing
      cmdUn.Caption = "Carica componente"
   Else
      Set X = CreateObject("Hacker.Application")
      cmdUn.Caption = "Scarica componente"
   End If
   Me.SetFocus
End Sub
```

Gerarchie di oggetti

Ci sono molti modi per organizzare oggetti, classi e gerarchie per le applicazioni ActiveX. In questo contesto, però, dovreste aver chiari due concetti particolarmente importanti: oggetti dipendenti e classi di collezione.

Oggetti dipendenti

Gli oggetti sono *dipendenti* se sono contenuti in altri oggetti. Le applicazioni client possono manipolare gli oggetti dipendenti, ma non possono crearli perché la proprietà Instancing dei moduli di classe degli oggetti dipendenti è impostata a 1-private o 2-PublicNotcreatable. Se il client non può crearli direttamente, qual è il meccanismo per creare gli oggetti dipendenti?
In questo caso, il componente ActiveX fornisce un metodo che crea l'oggetto dipendente. Tipicamente, il metodo si chiama Add o AddItem: il componente crea l'oggetto dipendente nel codice del metodo e restituisce al client un riferimento al nuovo oggetto appena creato.

Classi di collezione

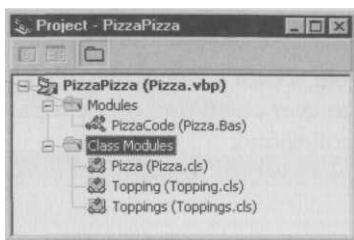
Una collezione è un oggetto che contiene un insieme di oggetti correlati (fate riferimento al Capitolo 14 per maggiori informazioni). Una classe di collezione è un modulo di classe che raccoglie oggetti simili che sono istanze di un'altra classe.

Una pizza virtuale

Per esempio, immaginate una classe Pizza istanziabile dall'esterno. Un oggetto collezione, Toppings, contiene tutte le possibili farciture per un possibile oggetto Pizza. Gli oggetti della classe non possono essere istanziati esternamente, ma possono solo essere creati utilizzando il metodo AddTopping della collezione Toppings della classe Pizza.
Questa gerarchia di oggetti richiede tre moduli di classe. Andando dal basso verso l'alto:

- Il modulo di classe Topping, che definisce un ingrediente per pizza per ogni istanza
- Il modulo di classe Toppings, che definisce una collezione di oggetti che contiene elementi Toppings
- Il modulo di classe Pizza, che definisce l'intera pizza con la sua farcitura (una collezione di oggetti Toppings)

potete vedere questi moduli di classe nel Project Explorer usando il progetto per il componente ActiveX di prova Pizza.Vbp, come si vede in Figura 23-19.

Figura 23.19
I moduli di classe del progetto Pizza servono come modelli per gli oggetti che vengono creati sulla base di queste classi.

La classe Topping contiene una proprietà, il nome dell'ingrediente:

```
Public name As String
```

Siccome gli oggetti della classe Topping sono dipendenti, la proprietà Instancing della classe dovrebbe essere impostata in modo da impedire la creazione dall'esterno. Visto che però le istanze della classe devono essere visibili all'esterno del contesto dell'applicazione ActiveX, la proprietà Instancing della classe va impostata a 2-PublicNotCreatable. Anche le istanze della classe Toppings sono dipendenti, e la proprietà Instancing del modulo di classe è impostata a 2-PublicNotCreatable. Questa classe fa da modello per la collezione di elementi Topping, e implementa un oggetto collezione privato:

```
Private ThisPizzaToppings As New Collection
```

I metodi della classe Toppings sono "involucri" attorno ai metodi dell'oggetto collezione di Visual Basic (si veda il paragrafo "Un modulo di classe è un involucro" nella parte precedente del capitolo). In altre parole, i metodi Toppings incapsulano chiamate di default ai metodi dell'oggetto collezione. Add Topping aggiunge un elemento Toppings alla collezione ThisPizzaToppings, mediante il metodo add dell'oggetto collezione e restituisce un riferimento al nuovo elemento Toppings. La proprietà NumberToppings interfaccia la proprietà Count della collezione per restituire il numero degli elementi presenti nella collezione definitiva dall'utente. Il metodo GetItem chiama il metodo Item della collezione ThisPizza Toppings per restituire un riferimento di oggetto all'elemento indicizzato nella collezione:

```
Public Function AddTopping(ByVal name As String)
   Dim ThisTopping As New Topping
   ThisTopping.name = name
   ThisPizzaToppings.Add ThisTopping
   Set AddTopping = ThisTopping
End Function ' Restituisce un riferimento al nuovo oggetto Topping

Proprietà Get NumberToppings() As Integer
   NumberToppings = ThisPizzaToppings.Count
End Property

Public Function GetItem(Which As Integer)
   Set GetItem = ThisPizzaToppings.Item(Which)
End Function  ' Restituisce un riferimento all'oggetto Topping
              ' selezionata in base al suo indice nella collezione
```

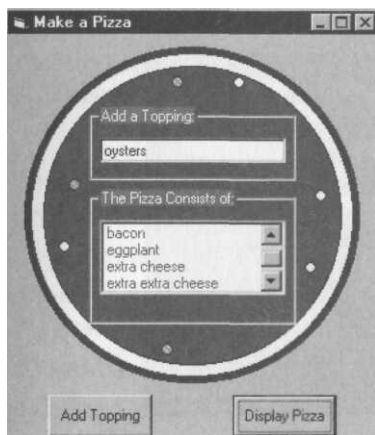
Infine il modulo di classe Pizza è una classe creabile esternamente (la sua proprietà Instancing è impostata a 5 - Multi Use). *Pizza* contiene una sola riga di codice:

```
Public Toppings As New Toppings
```

Questo significa che ogni oggetto Pizza creato sulla base del modulo di classe pizza avrà una propria collezione Toppings basata sulla classe di collezione Toppings. Un oggetto basato sulla classe di collezione Toppings può esistere solo come parte di un oggetto Pizza. Un oggetto Topping può essere creato solo chiamando il metodo AddTopping della collezione Toppings. Queste relazioni sono vincolate dalle impostazioni della proprietà Instancing dei moduli di classe.

Prepariamoci a cuocere un po' di pizze! Per usare il componente ActiveX Pizza, avviatelo come applicazione ActiveX utilizzando una Sub Main pro forma come abbiamo fatto per gli altri server. Il progetto client, ovviamente, si chiama PizzaCli.Vbp. Visto che la connessione tra client e server avviene per early binding (l'istruzione Dim X As New; vedi "Binding" in questo capitolo), il primo passo è verificare che il componente ActiveX *Pizza* sia selezionato nella finestra di dialogo *References.*

Il client di Pizza crea un oggetto della classe Pizza, che, a sua volta, istanzia un oggetto della classe di collezione Toppings. I metodi della collezione standard incapsulata nella classe Toppings sono usati per implementare la funzionalità visualizzata in Figura 23.20: aggiungere un ingrediente all'istanza della collezione Toppings, e visualizzare i nomi di tutti gli oggetti Topping presenti nella collezione.

Figura 23.20 *La proprietà Instancing dei moduli di classe permette di vincolare le gerarchie di oggetti.*

Il modulo del form per prima cosa istanzia un oggetto della classe Pizza, e crea automaticamente una nuova collezione Toppings per la classe:

```
Option Explicit
Dim X As New Pizza
```

La procedura "Add a Topping" chiama il metodo di AddTopping della classe di collezione:

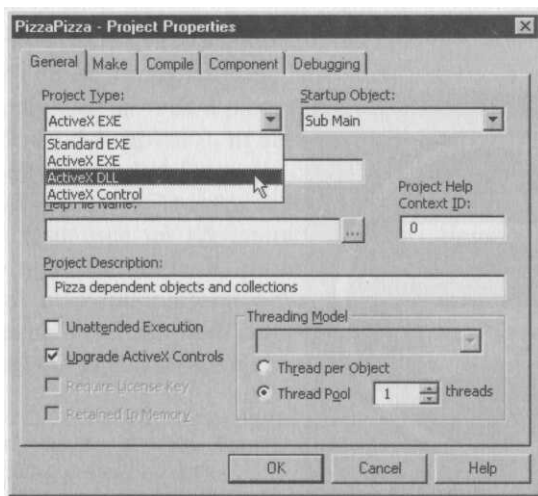
```
Private Sub cmdAdd_Click()
   X.Toppings.AddTopping txtTopping
End Sub
```

La procedura "The Pizza Consists of" usa la proprietà NumberToppings e il metodo GetItem della classe di collezione per aggiungere all'elenco i nomi di tutti gli ingre dienti presenti nella collezione:

```
Private Sub cmdDisplay_Click()
   Dim K As Integer
   Por K = 1 To X.Toppings.NumberToppings
      lstToppings.AddItem   X.Toppings.GetItem(K) .Name
   Next K
End Sub
```

Creare server in-process (DLL ActiveX)

Per creare server in-process (DLL ActiveX) si seleziona *ActiveX DLL* come tipo di progetto nella scheda *General* della finestra di dialogo *Project Properties* (come si vede nella Figura 23.21). La creazione di una DLL ActiveX assomiglia per molti versi alla creazione di un componente ActiveX indipendente.

Figura 23.21
L'opzione ActiveX DLL sotto Project Type permette di compilare un progetto come DLL ActiveX.

È possibile specificare ActiveX DLL *come tipo di progetto anche nella finestra di dialogo* New Project.

I server in-process non possono essere eseguiti indipendentemente. Poiché girano nello stesso processo dell'applicazione che li usa, normalmente risultano più veloci e consumano meno risorse rispetto ai server out-of-process (come le applicazioni ActiveX).
Ci sono tuttavia dei limiti al contenuto di un server in-process VB. Se riuscite a sopportarli, i server in-process sono la scelta migliore per progetti come:

aggiunte e wizard Visual Basic

moduli compilati per essere chiamati da client di automazione OLE come Excel

spezzare eseguibili di grandi dimensioni in moduli più piccoli

Vincoli sulle DLL in-Process

Le DLL ActiveX in-process devono sottostare ai seguenti vincoli:

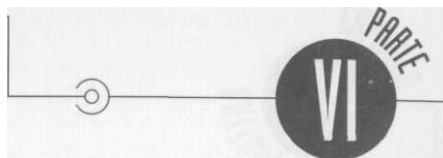
- Essere esclusivamente a 32-bit ed essere eseguite solo con sistemi operativi a 32-bit (Windows 95/98 e NT).
- Avere almeno un modulo di classe per cui la proprietà `Instancing` sia impostata a `5 - Multiuse`.
- Non visualizzare form non modali.
- Non utilizzare l'istruzione `End`, che nel caso dei server in-process provoca un errore di compilazione. (Notate che `End` non dovrebbe essere utilizzata neanche per i server out-of-process, perché di norma la durata della vita di un server dovrebbe essere controllata dal client.)

Riepilogo

Creare componenti ActiveX può essere decisamente stimolante, dal punto di vista informatico. Questo capitolo è stato dedicato all'esame degli strumenti, dei concetti e delle regole del gioco che vi servono per riuscire a sfornare applicazioni ActiveX funzionanti.

- Abbiamo visto come creare un'applicazione ActiveX.
- Abbiamo visto come istanziare gli oggetti basati su una classe.
- Abbiamo visto come chiamare i componenti ActiveX da applicazioni client.
- Abbiamo visto come impostare i riferimenti per i client usando la finestra di dialogo *References.*
- Abbiamo visto come usare le funzioni `CreateObject` e `GetObject`.
- Abbiamo definito le classi creabili dall'esterno.
- Abbiamo visto come creare una classe di collezione.
- Abbiamo visto come lavorare con le gerarchie di oggetti e gli oggetti dipendenti.
- Abbiamo visto come visualizzare form nelle applicazioni ActiveX.
- Abbiamo visto come lavorare con gli eventi dei moduli di classe.
- Abbiamo visto come impostare le opzioni per i progetti ActiveX.
- Abbiamo visto come registrare i componenti ActiveX.
- Abbiamo visto come gestire gli errori nei componenti ActiveX.

- Abbiamo visto come personalizzare la finestra di dialogo *Component Request Pending.*
- Abbiamo visto come usare gli strumenti per il controllo di versione degli ActiveX.
- Abbiamo visto come creare un oggetto Application.
- Abbiamo visto come e quando usare early binding e late binding.
- Abbiamo visto come preparare una pizza virtuale orientata agli oggetti e anche come mangiarla!

CREAZIONE DI CONTROLLI ACTIVEX

I CONTROLLI ACTIVEX

- Che cos'è un controllo?
- Progetti ActiveX Control
- Ciclo di vita del controllo
- Osservare il comportamento del controllo
- PropertyBag
- Controlli e container
- L'interfaccia del controllo
- Licenze per i controlli
- Necessità di una licenza per lo sviluppatore

I controlli ActiveX sono componenti ActiveX (un termine più datato per i componenti ActiveX è "Server OLE") che rispettano alcuni criteri addizionali. Per ottenere maggiori informazioni riguardo i componenti ActiveX, consultate il Capitolo 20 e il Capitolo 23, che forniscono le nozioni di base sui controlli ActiveX; è infatti importante comprendere questi concetti prima di iniziare la creazione di controlli ActiveX personalizzati.

Che cos'è un controllo?

I controlli pronti per la distribuzione sono eseguibili ActiveX compilati con estensione .Ocx. Tali eseguibili non sono in grado di funzionare autonomamente, ma possono interagire con l'ambiente di sviluppo in fase di progettazione. Quando l'applicazione che utilizza il controllo ActiveX viene eseguita, il controllo si comporta in maniera differente rispetto a quando viene visualizzato in fase di progettazione, e le specifiche delle attività runtime del controllo dipendono dal codice che lo sviluppatore ha associato ai membri del controllo stesso.

I membri del controllo sono le sue proprietà, i suoi metodi e i suoi eventi, e sono detti anche interfaccia *del controllo.*

È possibile interagire con il controllo in tre modi differenti:

- Progettando il controllo in ambiente VB
- Progettando un'applicazione con VB, o con un altro IDE, utilizzando il controllo
- Utilizzando il controllo in veste di componente di un'applicazione in fase di esecuzione

Queste tre modalità individuano tre categorie di utilizzatori:

- Chi progetta il controllo (sviluppatore)
- Chi utilizza il controllo per sviluppare un'applicazione (sviluppatore)
- Chi utilizza il controllo eseguendo un'applicazione (utente finale).

Le applicazioni Web eseguite attraverso Microsoft Internet Explorer 4 (e successivi) sono uno dei campi di utilizzo più importanti per i controlli creati con Visual Basic Per maggiori informazioni riguardo questo argomento, consultate il Capitolo 27.

I controlli ActiveX possono essere ospitati da differenti ambienti di sviluppo, come per esempio, Microsoft Access, Visual Basic, Delphi, Visual C++ e strumenti di sviluppo Web come Visual InterDev. Quando un ambiente di sviluppo ospita un controllo ActiveX all'interno di un progetto, esso viene aggiunto alla *Toolbox* (casella degli strumenti) relativa a quel particolare progetto. In VB è possibile aggiungere componenti alla Toolbox mediante la finestra di dialogo *Components,* come mostrato dalle Figure 24.1 e 24.2

Figura 24.1 *Aggiunta di un controllo ActiveX alla Toolbox RemoteData6.0).*

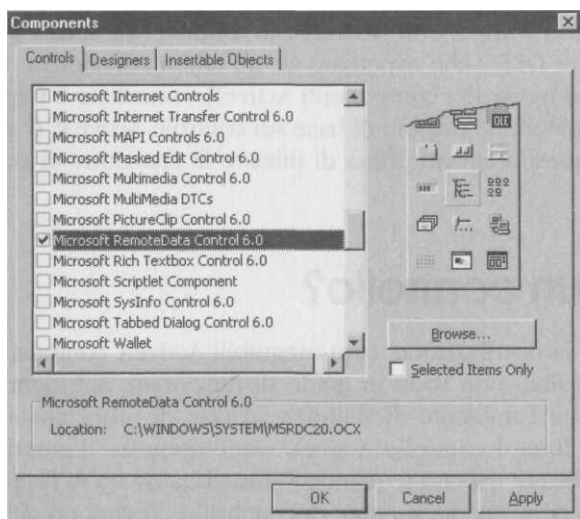
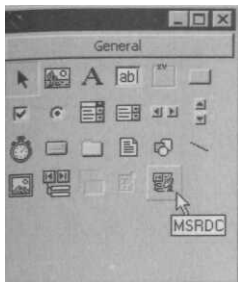



Figura 24.2 *Controllo RemoteData6.0 all'interno della Toolbox*

Il file diprogetto (. Vbp) viene utilizzato, tra le altre cose, per tenere traccia di quali controlli ActiveXsono stati aggiunti alprogetto VB. Per esempio, ecco leprime righe di un file diprogetto contenente il controllo RemoteData 6.0 :

```
Type=Exe
Form=Form1.frm
Object={F6125AB1-8AB1-11CE-A77F-08002B2F4E98}#2.0#0;     MSRDC20.0CX
Startup="Form1"
Command32=""
Name="Project1"
```

Per maggiori informazioni sul formato interno dei file di progetto VB, consultate il paragrafo del Capitolo 19 che descrive nel dettaglio la struttura di tali file.
Una volta aggiunto il controllo alla Toolbox, è possibile inserirlo in qualsiasi oggetto che sia in grado di fungere da *contenitore* di controlli ActiveX, ossia di *ospitarli.* L'esempio più comune di oggetto in grado di ospitare controlli è il form di Visual Basic, ma esistono anche altri oggetti (alcuni controlli, per esempio) in grado di fungere da contenitori di controlli. Per maggiori informazioni consultate il paragrafo che parla espressamente del rapporto tra controlli e container, più avanti in questo capitolo. Per aggiungere un controllo all'interno di un container è possibile procedere in due modi:

- Facendo doppio clic su un controllo all'interno della *Toolbox,* esso viene aggiunto all'interno dell'oggetto attivo.

  Selezionando il controllo all'interno della *Toolbox,* è possibile "disegnarne" il contorno all'interno di un oggetto container, e selezionarlo successivamente utilizzando lo strumento *Pointer* (Figura 24.3.).

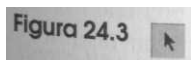



Figura 24.3

Una volta aggiunto il controllo al progetto, come mostrato nella Figura 24.4, è possibileaccedereaisuoimembriinmodalitàprogettazione.

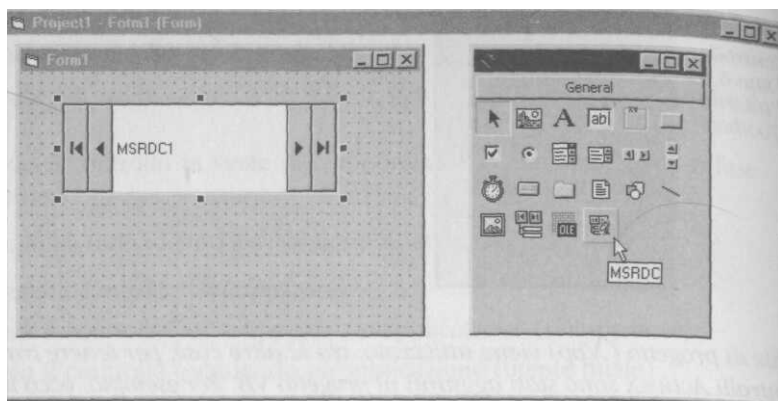
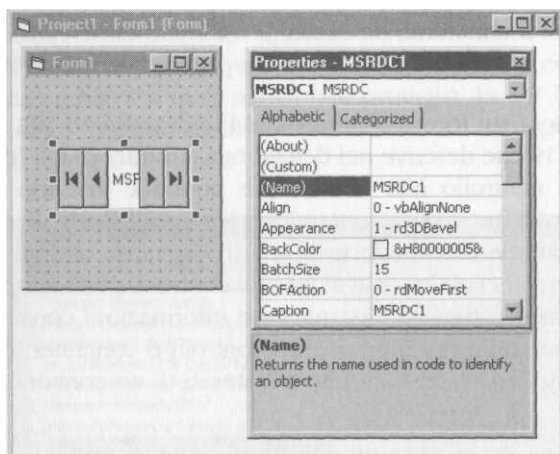
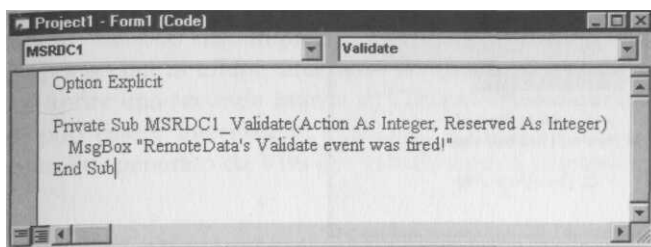
Figura 24.4
Aggiunta di un controllo al progetto all'interno di un form o di qualche altro contenitore.

È poi possibile utilizzare la finestra di dialogo *Properties* (mostrata nella Figura 24.5) per impostare le proprietà del controllo. È chiaramente possibile impostare e leggere le proprietà del controllo anche in fase di esecuzione, tramite il codice.

Figura 24.5
Utilizzo della finestra Properties per impostare le proprietà di un controllo ActiveX in fase di progettazione.

I metodi del controllo possono essere invocati solamente in fase di esecuzione, mentre gli eventi vengono automaticamente aggiunti all'insieme di procedure del progetto, per permettere di associare agli eventi stessi il codice personalizzato per la loro gestione. In Figura 24.6 viene aggiunto un comando MsgBox all'evento Validate del controllo RemoteData.

È importante distinguere tra gli eventi che vengono generati dal controllo egli eventi ricevuti dal controllo. Per una spiegazione riguardo questa distinzione, consultate il paragrafo che descrive il ciclo di vita del controllo, più avanti in questo capitolo.

Figura 24.6 Aggiunta del codice da eseguire in seguito a un evento.

L' interfaccia di progetto descritta finora è già familiare a tutti gli sviluppatori VB che abbiano utilizzato almeno una volta un controllo di terzi (e questo accade molto spesso, dato che un punto di forza degli ambienti di sviluppo come Visual Basic consiste proprio nella loro estendibilità). È possibile trovare informazioni supplementari sulla creazione di un'interfaccia per i propri controlli in un paragrafo apposito più avanti in questo capitolo, e nel paragrafo del Capitolo 25 che spiega nel dettaglio come creare tale interfaccia.

la possibilità di creare controlli ActiveX personalizzati senza abbandonare l'ambiente VB Cuna caratteristica già introdotta nella versione 5) è uno degli aspetti più importanti dello sviluppo Visual Basic. Un'implicazione consiste nel fatto che questa sua caratteristica cestina definitivamente l'idea che VB6possa essere solo un linguaggio "giocattolo". Inoltre, i controlli ActiveXpossono essere utilizzati anche per lo sviluppo di applicazioni Web (a patto che venga utilizzato un browser Explorer dotato di Visual Basic Virtual Machine, in breve VM).

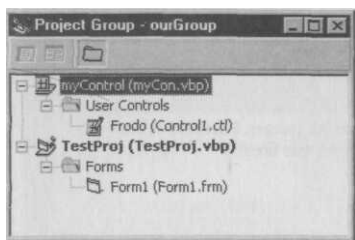
Progetti ActiveX Control

È possibile creare un nuovo progetto di controllo ActiveX scegliendo *New* dal menu *File* di VB e selezionando *ActiveX Control* come tipo di progetto.

Se l'opzione Prompt for Project *è stata impostata nella scheda* Environment *della finestra di dialogo* Options, *la creazione di un nuovo progetto viene avviata automaticamente a ogni apertura di VB.*

E inoltre possibile aggiungere un progetto ActiveX Control a un progetto preesistente (come, per esempio, un progetto Standard Exe, ovvero un eseguibile standard) scegliendo la voce *Add Project* dal menu *File* di VB. In alternativa, è possibile aggiungere il progetto Standard Exe a un progetto ActiveX Control preesistente. Una volta che i due progetti vengono aperti contemporaneamente in VB, viene a formarsi un Project Group, ovvero un gruppo di progetti, come mostrato nella Figura 24.7.

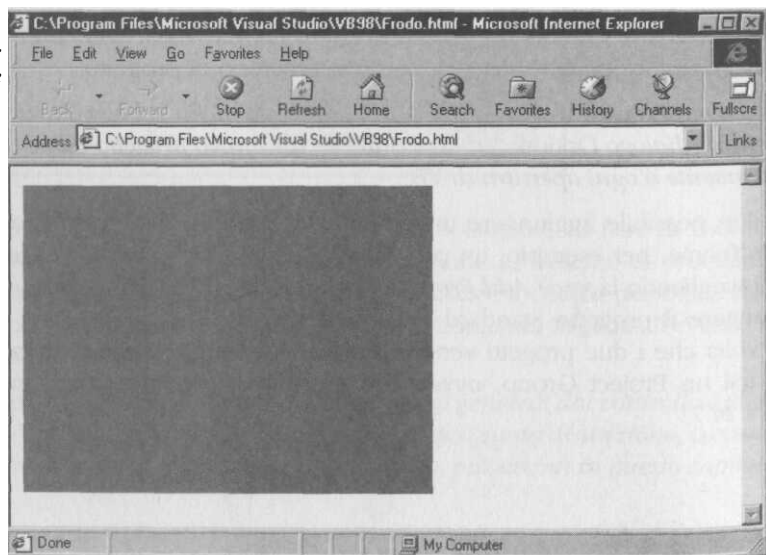
Figura 24.7

Project Group formato dai due progetti ActiveX Control e Standard Exe.

Un controllo ActiveX (per esempio un file .Ocx) non può essere eseguito autonomamente, anche se è stato compilato. In teoria potrebbe essere possibile effettuare dei test sul controllo compilandolo, aprendo un nuovo progetto VB e aggiungendovi il controllo attraverso la finestra di dialogo Component. *Bisognerebbe poi inserire il controllo in un form o in un contenitore, aggiungere codice di test e avviare il nuovo progetto per assicurarsi che il controllo funzioni.*

In pratica, è molto più semplice creare un Project Group che contenga il progetto del controllo e un progetto eseguibile standard pensato per mettere alla prova il controllo. Supponendo che il progetto del controllo sia stato salvato e costruito correttamente, il nuovo controllo ActiveX apparirà automaticamente nella Toolbox del progetto di test.

Una limitazione del Project Group in Visual Basic 6 consiste nel fatto che quando si utilizza un Project Group in fase di progettazione, l'ambiente di sviluppo utilizza sempre Internet Explorer (invece di un form Visual Basic) come contenitore. Nella Figura 24.8 viene mostrata l'esecuzione di un Project Group contenente un controllo utente di nome "Frodo".

Figura 24.8

Internet Explorer come contenitore durante l'esecuzione di un Project Group in VB6.

In conclusione, per effettuare dei test di un controllo ActiveX mediante un form VB come contenitore, sarebbe comunque necessario compilare il controllo in un file .Ocx. Ciò comporta che, in effetti, una volta compilato il controllo, sarebbe altrettanto semplice aprire una seconda istanza di VB6 per effettuarne il test.
In ogni caso, può essere interessante analizzare il codice HTML (che si basa sul CLSID del controllo) generato da VB6 per visualizzare la pagina mostrata in Figura 24.8:

```
<HTML><BODY><OBJECT classid="clsid:BA758830-D6A4-11D1-
   B853006008A093F0">
</OBJECT></BODYx/HTML>
```

I codici CSLID sono descritti nel Capitolo 20.
Per maggiori informazioni riguardo i Project Group e i diversi tipi di progetti di VB6, consultate invece il paragrafo del Capitolo 3 che spiega come lavorare con i file sorgente di VB.

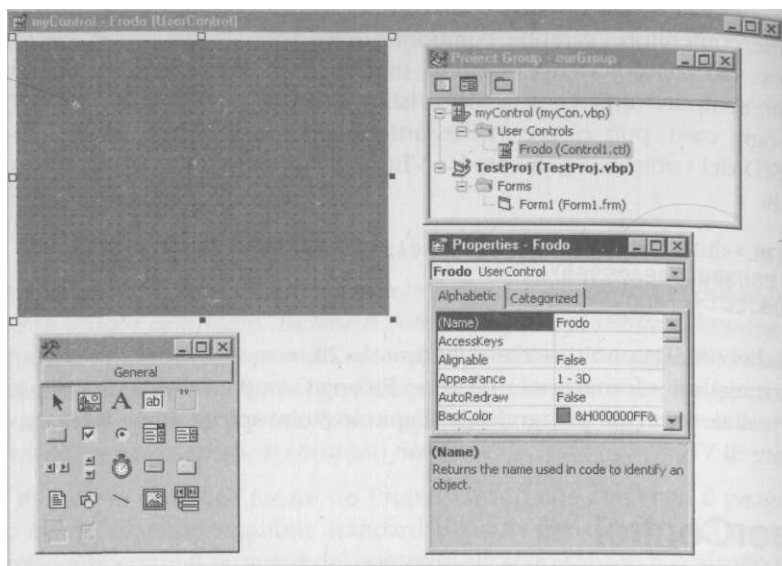
UserControl

Un oggetto UserControl sta a un progetto ActiveX Control come un form sta a un progetto Standard Exe. Per default, ogni nuovo progetto ActiveX Control viene creato con, al suo interno, un oggetto UserControl, così come un progetto Standard Exe viene creato con un oggetto form.
È possibile utilizzare tutti gli strumenti standard di VB, come, per esempio, la finestra di dialogo *Properties,* per modificare l'aspetto di un oggetto UserControl, come mostrato in Figura 24.9. (È da notare che non è necessario che l'oggetto UserControl sia visibile in fase di esecuzione.)
I progetti ActiveX Control possono contenere altri tipi di moduli oltre a UserControl (file .Ctl), come form (file .Frm), moduli standard (file .Bas) e moduli di classe (file .Cls). La comprensione dei moduli di classe, e del modo in cui vengono utilizzati in VB, è particolarmente importante per poter progettare in maniera efficace controlli ActiveX.

Classi

I moduli di classe (descritti in dettaglio nel Capitolo 14) contengono proprietà, metodi e eventi. Le proprietà vengono implementate dalle procedure Property, o attraverso delle variabili. I metodi vengono implementati attraverso funzioni Public. Gli eventi vengono dichiarati con la parola chiave Event e poi generati al momento adatto. Per esempio:

```
Event RedAlert (EnterpriseInDanger As Boolean, _
   WhoCausedIt As String)

 If AppropriateCircumstances Then
   RaiseEvent RedAlert(True, "The Borg")
End if
```

Figura 24.9
Per agire su UserControl e form si utilizzano gli stessi strumenti.

I moduli di classe non hanno una rappresentazione propria sullo schermo; ciò nonostante, si rende spesso necessario aggiungerne ai componenti. Nelle versioni più vecchie di Visual Basic, i moduli di classe venivano utilizzati per incapsulare le funzionalità interne dell'applicazione. Ogni modulo di classe pubblico aggiunto a un controllo è assimilabile a uno "stampo" che può servire per creare un oggetto che farà parte del controllo.

Il nome che viene fornito al modulo di classe viene combinato con il nome del controllo per ottenere *un progID* (programmatic ID) per la classe, ossia un identificativo. Un esempio di ProgID per una nuova classe potrebbe essere `myTool.myNewClass`.

La proprietà `Instancing` di ogni modulo di classe definisce come sia possibile accedervi. Per i progetti ActiveX Control, la proprietà `Instancing` del modulo di classe deve essere impostata a `Private`, `PublicNotCreatable`, `Multiuse` o `GlobalMultiUse`. `Private` significa che le applicazioni esterne non potranno accedere alle informazioni della classe e non saranno in grado di creare oggetti basandosi su di essa. `PublicNotCreatable` significa invece che le applicazioni esterne saranno in grado di accedere solamente alle istanze della classe create dal controllo. Gli oggetti che si basano su un modulo di classe `PublicNotCreatable` vengono anche detti *"dependent",* ossia dipendenti, in quanto devono essere creati dal componente genitore prima di essere utilizzati. Se desiderate che le applicazioni esterne siano in grado di creare oggetti dipendenti, dovete fornire un metodo del componente (come, per esempio, `Add`) che crei un'istanza dell'oggetto dipendente.

La possibilità di impostare a Multiuse o GlobalMultiUse *l'istanziazione di un modulo di classe è stata introdotta nella versione 6 di VB. I moduli di classe impostati in questa maniera, permettendo l'istanziazione dall'esterno, hanno una nuova proprietà,* Persistable, *chepuò essere impostata solo infase di esecuzione. Il valoreprede/mito,* vbNotPersistable *(ossia O) significa che l'oggetto creato dal modulo di classe non è persistente. D'altro canto, se la proprietà è impostata a* vbPersistable *(ossia 1), l'oggettopuò scrivere e leggere dati tra un 'istanza e l'altra. Una volta che la proprietà* Persistable *è stata impostata a* vbPersistable, *gli eventi* initProperties, ReadProperties *e* WriteProperties *e il metodo* PropertyChanged *vengono aggiunti alla classe. Il significato di questi membri della classe viene approfondito nel contesto di un generico modulo del controllo, più avanti in questo capitolo. Per saperne di più sulla persistenza, consultate invece il paragrafo cheparla della* PropertyBag, *sempre in questo capitolo.*

Creazione di pacchetti di controlli ActiveX

Una ragione della possibilità di creare eseguibili standard in VB6 ha a che fare con lo "scope", ossia la *visibilità* o *l'ambito* d'azione, dei controlli ActiveX inclusi nel progetto. Infatti, se lo si preferisce, invece di distribuire il proprio controllo ActiveX in un file .Ocx separato, *è* possibile includerne il codice sorgente direttamente nell'eseguibile del progetto standard compilando il Project Group invece di compilare il controllo separatamente. L'effetto indiretto di questo tipo di approccio consiste nel fatto che il controllo ActiveX diventa privato all'eseguibile, e questa conseguenza può produrre effetti graditi ed effetti sgraditi allo stesso tempo.
Un controllo ActiveX di VB6 *è* contenuto in un progetto ActiveX Control contenente almeno un modulo sorgente UserControl (salvato con l'estensione .Ctl). La scelta di compilare il progetto in un unico eseguibile oppure di compilare il controllo in un file .OCX separato che deve poi essere distribuito assieme all'applicazione, fa parte di un contesto più ampio riguardante i *pacchetti di distribuzione.*

Un progetto ActiveX Controlpuò contenere più di un modulo UserControl *epiù di un controllo ActiveX.*

I controlli Public sono controlli che possono essere utilizzati anche da altre applicazioni, e devono essere compilati all'interno di un progetto ActiveX Control (e la proprietà Public dell'oggetto UserControl deve essere impostata a True), ottenendo un file .Ocx. Allo stesso modo, un controllo può essere reso privato impostando a False la proprietà Public dell'oggetto UserControl relativo; in questo nodo, una volta compilato il progetto, i controlli privati potranno essere utilizzati solo all'interno del progetto nel quale sono stati compilati. La proprietà Public dell'oggetto UserControl non può essere impostata a True se l'oggetto non si trova in un progetto ActiveX Control. Se uno dei controlli di un progetto ActiveX Control verra utilizzato solamente da altri controlli dello stesso progetto, la sua proprietà **Public** può essere impostata a False; in questo modo le applicazioni esterne non saranno in grado di accedervi, mentre gli altri controlli del progetto potranno farlo.

Modifica del pacchetto

È possibile modificare facilmente il modo in cui il controllo viene inserito nel pacchetto spostandone i file sorgente in un progetto di tipo differente. Per esempio, se avete creato alcuni controlli privati che fanno parte di un progetto e desiderate renderli disponibili alle applicazioni esterne, potete aggiungere i file .Ctl a un nuovo progetto ActiveX Control, e compilarli in un nuovo controllo Ocx.
Se, al contrario, non desiderate distribuire un componente compilato aggiuntivo potete spostare i file .Ctl del progetto ActiveX Control nel progetto dell'applicazione; quando l'applicazione verrà compilata, il codice di gestione del controllo verrà compilato al suo interno. I vantaggi di includere il codice sorgente del controllo nell'eseguibile dell'applicazione sono :

- L'eliminazione della necessità di distribuire un file .Ocx separato.
- La semplificazione della fase di test del controllo, in quanto bisogna preoccuparsi solamente di come il controllo viene utilizzato dalla propria applicazione, e non di come potrebbe essere utilizzato da applicazioni esterne.
- L'eliminazione della necessità di distribuire nuove versioni aggiornate del controllo, in quanto la distribuzione avviene automaticamente distribuendo una nuova versione dell'applicazione.

Ecco invece gli svantaggi di includere i controlli nell'eseguibile dell'applicazione:

- Se si desidera aggiornare il controllo (oppure si scopre un bug nel controllo che deve essere eliminato) si rende necessario ricompilare e ridistribuire l'intera applicazione.
- Se gli stessi controlli venissero utilizzati da più applicazioni, il codice relativo dovrebbe essere distribuito più volte (all'interno di ogni applicazione), dato che le applicazioni non possono condividerlo, portando a un aumento dello spazio necessario sulle macchine destinatane.
- Il controllo della versione diventa difficoltoso, perché il codice sorgente utilizzato dalle varie applicazioni subisce inevitabilmente delle modifiche; questo comporterebbe inoltre una maggiore difficoltà nell'appurare quale particolare versione del controllo viene utilizzata da una particolare applicazione.
- Diventa molto più difficile condividere codice sorgente con gli altri programmatori e creare e distribuire un'interfaccia standard.
- Distribuendo codice sorgente invece che file .Ocx compilati, si perde la proprietà e la segretezza del codice sorgente stesso.

Ciclo di vita del controllo

È importante comprendere che le istanze dei controlli vengono continuamente create *e* distrutte. Questo non accade, invece, per le applicazioni basate su form. In tali applicazioni, le form vengono create e distrutte un numero limitato di volte durante una sessione di lavoro.

In Visual Basic 6, i moduli di classe con istanziazione pubblica impostati come persistenti assistono a tutto il ciclo di vita e a tutti gli eventi del controllo.

I seguenti eventi causano la creazione e la distruzione di un controllo:

Apertura e chiusura del contenitore che ospita il controllo

Aggiunta e rimozione del controllo all'interno del contenitore

Esecuzione del progetto che contiene il controllo

La creazione e la distruzione di un controllo coincidono con la creazione e la distruzione del modulo UserControl che contiene il codice del controllo (e di tutti i form e i moduli associati). Ecco un esempio del ciclo di vita di un controllo UserControl (non ancora compilato) in ambiente di sviluppo VB.

- Lo sviluppatore che utilizza il controllo lo aggiunge all'interno di un form. Viene creata un'istanza del controllo nel form.

Per aggiungere un controllo a un form quando il controllo fa parte di un progetto ActiveX Control in modalità di progettazione, è necessario chiudere la finestra di progetto dello UserControl *relativo al controllo.*

- L'evento Initialize del controllo viene generato.
- Viene generato uno dei due eventi seguenti: InitProperties, se l'evento Initialize era stato innescato dall'inserimento di una nuova istanza del controllo in un form, oppure ReadProperties, se viene riaperto un form già salvato (in cui era stato precedentemente incluso il controllo).

InitProperties *imposta le proprietà del controllo ai rispettivi valori predefiniti, mentre* ReadProperties *legge i valori delle proprietà memorizzati precedentemente all'interno del contenitore. Il meccanismo per la memorizzazione e la lettura dei valori delle proprietà (a patto che il controllo sia persistente) è descritto in dettaglio nel paragrafo che parla dell'oggetto* PropertyBag, *più avanti in questo capitolo.*

- A questo punto vengono generati gli eventi relativi alla visualizzazione del controllo, ovvero:
  - Resize, che causa il ridimensionamento del controllo alle dimensioni che gli erano state assegnate all'interno del form
  - Show, che visualizza il controllo
  - Paint, che scatta dopo che il codice dell'evento paint del controllo è stato eseguito.

Eventi: È meglio dare o ricevere ?

Come già accennato in precedenza, è importante distinguere tra gli eventi generati dal controllo e quelli ai quali il controllo può rispondere. È possibile dare la seguente interpretazione delle due categorie: gli eventi generati dal controllo rappresentano una possibilità per lo sviluppatore di svolgere operazioni in risposta a un evento mentre gli eventi ricevuti dal controllo rappresentano la possibilità, per il controllo dì svolgere operazioni.
A questo proposito, noterete come la gestione degli eventi nella finestra *Code Editar* di VB sia profondamente diversa a seconda che il controllo si trovi in un proprio progetto oppure sia stato inserito in un container di un progetto di test.

Supponiamo ora di avviare l'esecuzione del progetto contenente il Form che a sua volta contiene il controllo. L'IDE Visual Basic chiude il form, generando l'evento WriteProperties, che salva i valori correnti delle proprietà del controllo. Successivamente viene innescato l'evento Terminate del controllo e l'istanza del controllo vienedistrutta.
A questo punto, durante il processo di creazione di un'istanza runtime del form, viene creata anche un'istanza runtime del controllo, e viene eseguita la sequenza di eventi già descritta. Come è facile intuire, quando il form runtime viene chiuso e l'IDE torna in modalità di progettazione, viene nuovamente innescata la sequenza di eventi che determina la distruzione dell'istanza del controllo e, quando il form viene riaperto in modalità progettazione, vengono nuovamente innescati gli eventi per la sua creazione.
A breve vedremo come sia possibile aggiungere comandi Debug.Print agli eventi del controllo per poterne tenere sotto controllo il ciclo di vita. Ma prima è opportuno fare alcune considerazioni sugli eventi del controllo.

Osservazione del comportamento del controllo

Sicuramentepuò risultare molto interessante assistere ed essere testimoni del verificarsi degli eventi durante il ciclo di vita di un controllo. Per farlo, create prima di tutto un progetto ActiveX Control. (L'esempio si trova sul CD-ROM con il nome di myCon.vbp.) Se lo desiderate, potetepersonalizzare lo UserControl *contenuto nel progetto myControl, magari cambiandone il nome in "Frodo" e modificandone il colore difondo in rosso, impostando la proprietà* BackColor. *Aprite ora il Code Editarper lo* UserControl, *e aggiungete il codice necessario agli eventifondamentali del ciclo di vita del controllo, come mostrato nelListato 24.1.*

Listato 24.1 *Il ciclo di vita di un controllo.*

```
Private Sub UserControl_Initialize()
    Debug-Print "Svegliami, scuotimi! Inizializza..."
End Sub
Private Sub UserControl_InitProperties()
    Debug-Print"Per la prima volta! InitProperties... "
End Sub
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    Debug.Print "Ormai siamo esperti! ReadProperties... "
End Sub


Private Sub UserControl_Terminate()
    Debug.Print "Vivrete senza di me! Terminate..."
End Sub
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    Debug.Print "Salviamo le proprietà! WriteProperties"
End Sub
```

Quando avrete finito di sperimentare, chiudete l'oggetto `UserControl`.

Osservando la Toolbox di TestProj prima di chiudere l'oggetto `UserControl`, *noterete la presenza dell'icona di myControl. Ciononostante, l'icona è in grigio, il che indica che il controllo è disattivato e non è possibile inserirlo in un contenitore.*

Selezionate ora *Add Project* dal menu *File* per creare un nuovo progetto *Standard Exe.* Il nuovo controllo farà già parte della Toolbox, come mostrato in Figura 24.10.

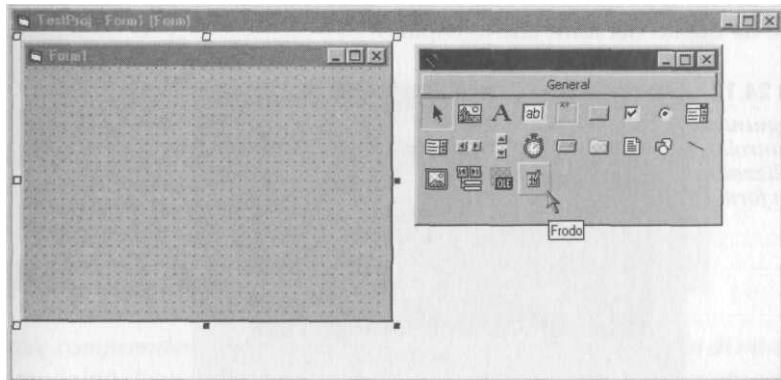
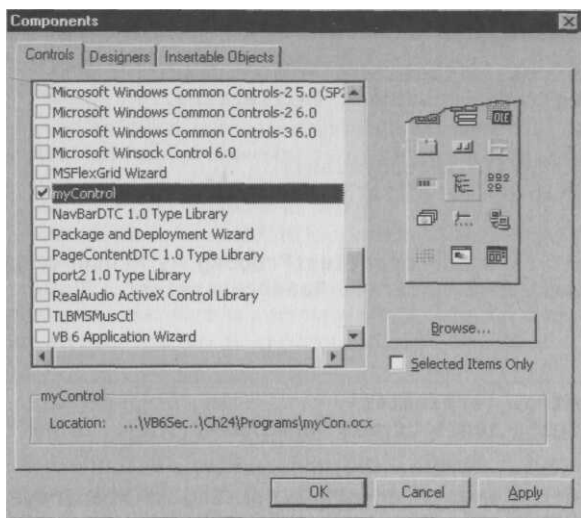
Figura 24.10 *L'icona del controllo viene automaticamente aggiunta ai nuovi progetti.*

Come già spiegato in precedenza, quando si avvia l'esecuzione di un Project Group, il controllo viene automaticamente aperto da Internet Explorer, che funge da contenitore. Se però si desidera osservare il comportamento del controllo all'interno di un form VB, *si rende necessario compilare prima il controllo, avviare un'altra istanza di VB, creare un nuovo progetto Standard Exe e utilizzare la finestra di dialogo* Components *per aggiungere il nuovo strumento alla Toolbox, come mostrato nella Figura 24.11.*

Figura 24.11 *Utilizzo della finestra Components per aggiungere il controllo personalizzato al progetto.*

La lista alfabetica mostrata dalla finestra di dialogo *Components* presenta il nome impostato nel campo *Project Name* della scheda *General* della finestra di dialogo *Project Properties* del progetto ActiveX Control. È possibile avere più di un controllo con lo stesso nome; ciò può accadere quando esiste un'istanza compilata di un controllo e il progetto dello stesso controllo è anche aperto in modalità progettazione nell'ambiente di sviluppo VB.

Fate doppio clic sull'icona del controllo per aggiungerlo al form. La Figura 24.12 mostra il controllo appena aggiunto, circondato dai simboli di ridimensionamento, all'interno della finestra *Form1*. (Il controllo ha un aspetto radicalmente diverso al resto del form in quanto il suo colore di sfondo è stato impostato in maniera differente da quello del form che lo ospita.)

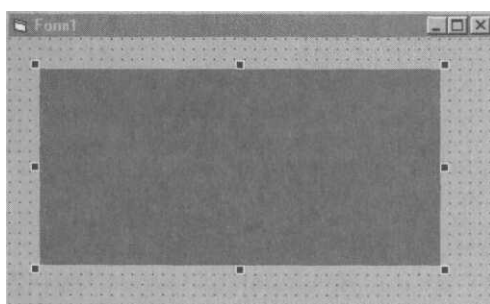
Figura 24.12 *Aggiunta di un controllo personalizzato a un form.*

Assicuratevi che la finestra *Immediate* sia visibile, in modo che sia possibile osservare i messaggi generati dai comandi di Debug mostrati nel Listato 24.1. Ora avviate il progetto. Come si vede nella Figura 24.13, vengono generati gli eventi `InitProperties` e `ReadProperties` del controllo. (Se il controllo fosse basato su un modulo di classe persistente, `InitProperties` non verrebbe innescato in quanto il controllo avrebbe già dei valori di proprietà differenti da quelli predefiniti.)

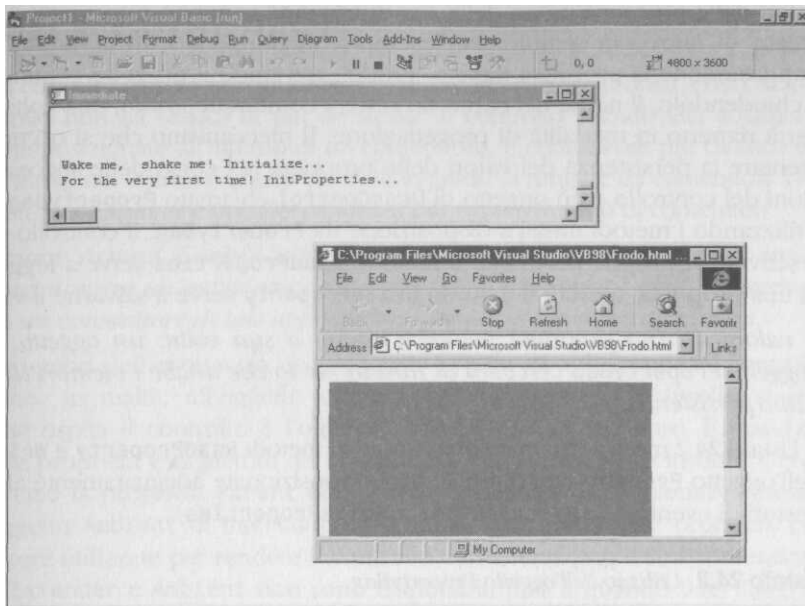
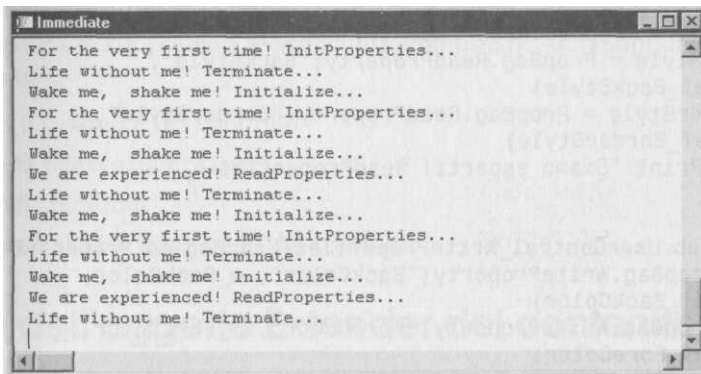
Figura 24.13
Innesco degli eventi di creazione del controllo.

Se si arresta l'esecuzione del progetto e lo si avvia nuovamente, ci si accorge che gli eventi di distruzione e di creazione del controllo vengono innescati più volte, come mostrato in Figura 24.14.

Figura 24.14
Eventi di distruzione e creazione quando si esce dalla modalità runtime e si entra in quella di progettazione.

È fondamentale comprendere che esistono due differenti versioni del ciclo di vita del controllo: in modalità di progettazione, quando il controllo viene posizionato ed eliminato dal form, e in modalità esecuzione, quando viene creato e distrutto.

PropertyBag

Come è possibile vedere osservando il ciclo di vita del controllo, è spesso fondamentale che le proprietà del controllo siano persistenti. Ciò significa infatti che i

valori delle proprietà prima della sua distruzione vengono ripristinati quando viene creato di nuovo in seguito. Per esempio, aggiungendo un controllo a un form impostandone la proprietà Name a "Frodo", avviando il progetto contenente il form, e chiudendolo, il nome del controllo resterà comunque *Frodo,* una volta che il form verrà riaperto in modalità di progettazione. Il meccanismo che si occupa di implementare la persistenza dei valori delle proprietà nel corso delle successive invocazioni del controllo è un oggetto di UserControl chiamato PropertyBag. Utilizzando i metodi messi a disposizione da PropertyBag, il controllo può leggere e scrivere le proprie proprietà. Il metodo ReadProperties serve a leggere il valore di una proprietà, mentre il metodo WriteProperty serve a salvarne il valore.

// valore di una proprietà potrebbe essere, a sua volta, un oggetto. In tal caso l'oggetto PropertyBag *cercherà difare in modo che anche i membri di tale oggetto siano persistenti.*

Il Listato 24.2 mostra chiamate di esempio ai metodi ReadProperty e WriteProperty dell'oggetto PropertyBag di un controllo (posizionate adeguatamente all'interno dei gestori di eventi di ReadProperties e WriteProperties).

Listato 24.2 *Utilizzo dell'oggetto PropertyBag*

```
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
   m_BackColor=PropBag.ReadProperty("BackColor",_
      m_def_BackColor)
   m_ForeColor = PropBag.ReadProperty("ForeColor", _
      m_def_ForeColor)
   m_Enabled = PropBag.ReadProperty("Enabled", m_def_Enabled)
   Set Font = PropBag.ReadProperty("Font")
   m_BackStyle=PropBag.ReadProperty("BackStyle",_
      m_def_BackStyle)
   m_BorderStyle = PropBag.ReadProperty("BorderStyle", _
      m_def_BorderStyle)
   Debug.Print "Siamo esperti! ReadProperties..."
End Sub

Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
   Call PropBag.WriteProperty("BackColor", m_BackColor, _
      m_def_BackColor)
   CallPropBag.WriteProperty("ForeColor", m_ForeColor, _
      m_def_ForeColor)
   CallPropBag.WriteProperty("Enabled", m_Enabled, _
      m_def_Enabled)
   Call PropBag.WriteProperty("Font", Font)
   Call PropBag.WriteProperty("BackStyle", m_BackStyle, _
      m_def_BackStyle)
   Call PropBag.WriteProperty("BorderStyle", m_BorderStyle, _
      m_def_BorderStyle)
   Debug.Print "Salviamo le proprietà! WriteProperties"
End Sub
```

Controlli e contenitori

Così come il concetto di isola non ha senso senza l'acqua che la circonda, così l'istanza di un controllo non ha senso, di per se stessa. Il controllo prende vita solamente quando viene posizionato all'interno di un contenitore. Il contenitore più classico è il form Visual Basic, ma esistono anche controlli in grado di fungere da contenitori. Una finestra di Internet Explorer è uno dei candidati più recenti in fatto di contenitori.

Un'applicazioneActiveX Control non può essere eseguita direttamente. È infatti necessario aggiungere un 'applicazione Standard Exe al Project Group e inserire il controllo in un contenitore di tale applicazione per poterlo mettere alla prova.

Alcuni dei membri dell'interfaccia che vengono presentati agli utenti del controllo appartengono, in realtà, all'oggetto contenitore che ospita il controllo stesso; l'oggetto che ospita il controllo è l'oggetto *Extender* del contenitore. È possibile accedere alle proprietà e ai metodi del contenitore nel quale è stato inserito il controllo attraverso la proprietà Parent dell'oggetto Extender del contenitore stesso. Inoltre, l'oggetto Ambient di UserControl contiene informazioni e proprietà che possono essere utilizzate per rendere il controllo consistente con il suo contenitore. Gli oggetti Extender e Ambient non sono disponibili fino a quando UserControl non viene posizionato all'interno di un contenitore, e i suoi eventi InitProperties e ReadProperties non sono stati innescati in seguito all'evento Initialize.
Alcune caratteristiche dei controlli ActiveX richiedono un supporto da parte del contenitore nel quale vengono inseriti, e non tutti gli oggetti contenitore sono in grado di fornire supporto a tutte le caratteristiche disponibili. Ciò significa che, a seconda del tipo di contenitore, alcune caratteristiche potrebbero non essere disponibili.
I form Visual Basic supportano le seguenti caratteristiche, non supportate da molti altri contenitori:

- Sfondo trasparente del controllo
- Proprietà ControIcontainer
- Controlli allineabili
- Form non modali visualizzati dal controllo

Utilizzo dell'oggetto Extender del contenitore

Posizionando il controllo personalizzato all'interno di un contenitore come, per esempio, un form, e osservando le sue proprietà tramite la finestra di dialogo *Properties,* potrete rintracciare numerose proprietà che non avete definito; tali proprietà sono le proprietà dell'oggetto Extender fornite dal contenitore, anche se all'utente finale appaiono semplicemente come un'estensione senza soluzione di continuità del controllo. È possibile utilizzare le proprietà dell'oggetto Extender del contenitore per impostare le proprietà del controllo. Un esempio tipico è rappresentato dalle proprietà Caption e Name, i cui valori predefiniti vengono impostati dall'oggetto contenitore in base ai controlli preesistenti Solitamente il loro valore predefinito consiste nel nome del controllo seguito da un numero progressivo che rappresenta il numero di istanze di tale controllo all'interno del contenitore. In Tabella 24.1 viene fornito

l'elenco di tutte le proprietà dell'oggetto Extender che tutti i container devono fornir in base alle specifiche ActiveX.

Tabella 24.1 *Proprietà obbligatorie dell'oggetto Extender del contenitore.*

Proprietà	Tipo di dato	Accesso	Significato
Name	String	Lettura	Il nome che l'utente assegna all'istanza del controllo
Visible	Boolean	Lettura/ Scrittura	Indica se il controllo è visibile
Parent	Object	Lettura	Restituisce l'oggetto che contiene il controllo come, per esempio, la form Visual Basic
Cancel	Boolean	Lettura	True se il controllo è il pulsante Cancel del contenitore
Default	Boolean	Lettura	True se il controllo è il pulsante predefinito del contenitore

In realtà non tutti i contenitori mettono a disposizione queste proprietà. È quindi importante implementare la gestione degli errori per assicurarsi che le proprietà dell'Estender esistano realmente, quando le si utilizza.

Internet Explorer, che con VB6 diventa di fatto il contenitore di default, non supporta molte delle proprietà appena elencate (compresa Parent). Ciò significa che, come minimo, il codice deve includere comandi On Error Resume Next prima di ogni invocazione all'oggetto Extender del contenitore, altrimenti il codice potrebbe causare l'interruzione dell'esecuzione se il contenitore non dovesse supportare la proprietà a cui si fa riferimento.
Molti contenitori, invece, mettono a disposizione proprietà supplementari rispetto a quelle elencate in Tabella 24.1, come, per esempio, Left, Top, Width e Height.

Per manipolare la visibilità del proprio controllo in fase di esecuzione (o per permettere a un programmatore che usa il controllo di fare lo stesso), è bene non utilizzare la proprietà Visible *dell'*Extender, *ma utilizzare invece la proprietà* InVisibleAt - Runtime *dell'oggetto* UserControl.

È inoltre importante tenere a mente che, se l'oggetto Extender del contenitore e il controllo che ospita hanno una proprietà con lo stesso nome, la proprietà dell'oggetto Extender ha la precedenza.

La proprietà UserMode dell'oggetto Ambient

È importante sottolineare che la proprietà UserMode dell'oggetto Ambient permette all'istanza del controllo di determinare se si trova in modalità di progettazione (UserMode = False) o in modalità di esecuzione.

Una semplice regola che permette di tenere a mente il significato della proprietà UserMode *consiste nel ricordare che, in fase di progettazione, la persona che lavora sul controllo è lo sviluppatore, non l'utente finale; quindi il controllo non è* ***utilizzato*** *dall'utente* (UserMode = False).

Un esempio di utilizzo della proprietà UserMode può essere la definizione di una Captiondifferente per il contenitore in seguito all'evento di Resize del controllo, a econda che sia il programmatore o l'utente finale a innescarlo, come mostrato nel Listato 24.3.

Listato 24.3 *Esempio di utilizzo della proprietà UserMode dell'oggetto Ambient.*

```
Private Sub UserControl_Resize()
   On Error Resume Next
   If Ambient.UserMode Then
      Extender.Parent.Caption = "myTool says," + Chr(34) +_
         "Hi, end user!" + Chr(34)
   Else
      Extender.Parent.Caption = "Developer, thanks for" + _
         " All the fish and for using myControl!"
   End If
End Sub
```

In Figura 24.15 viene mostrata un'istanza di myControl all'interno di un form in modalità di progettazione. La Caption dedicata allo sviluppatore appare non appena il controllo viene posizionato nel container. Invece, nella Figura 24.16 vengano mostrati lo stesso controllo e il relativo form in modalità esecuzione (con la Caption orientata all'utente finale).

Figura 24.15 *Esempio della Caption della form in fase diprogettazione...*

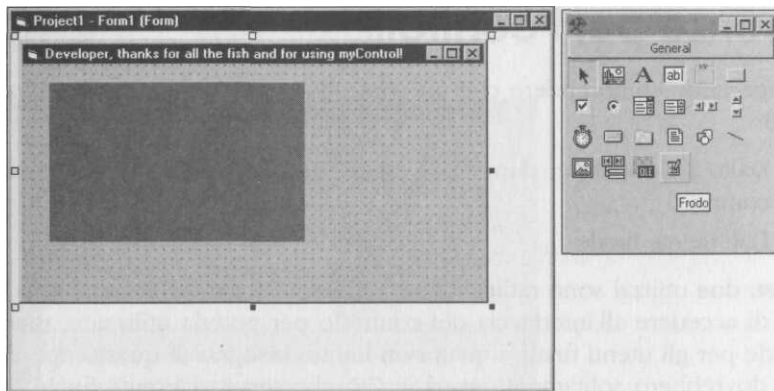
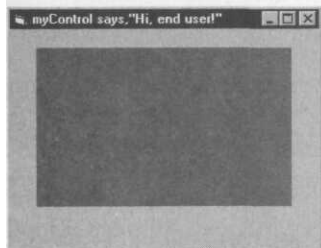



Figura 24.16 *..e infase di esecuzione*

L'interfaccia del controllo

Una volta che il controllo è stato inserito nel contenitore, è possibile esplorarne l'*interfaccia,* che è rappresentata dalle sue proprietà esposte, dai suoi metodi e dai suoi eventi. Nel corso del Capitolo 25 viene descritto come progettare e implernentare gli elementi dell'interfaccia di un controllo. Nel frattempo, ecco un riassunto dei concetti chiave che la riguardano.

- Un controllo ActiveX espone le proprie proprietà, i propri metodi e i propri eventi seguendo uno standard. Ciò significa che ogni applicazione (da Visual Basic a Excel ai linguaggi di scripting Web) che è in grado di comprendere questo standard può comunicare con il controllo e manipolarlo.
- Le *proprietà* servono per memorizzare e leggere informazioni memorizzate all'interno del controllo.
- I *metodi* e le funzioni, quando eseguiti, fanno svolgere operazioni al controllo.
- Gli *eventi* sono procedure del codice (spesso indicate con il termine "event handler" ovvero gestori di eventi) di cui un oggetto innesca l'esecuzione quando si verificano particolari condizioni, e rappresentano l'opportunità per l'utente di aggiungere codice proprio all'interno dei progetti che contengono il controllo.

Licenze per i controlli

È importante comprendere che un controllo ActiveX può essere utilizzato in due modi:

- Dallo sviluppatore, durante la creazione di applicazioni che utilizzano il controllo
- Dall'utente finale

Questi due utilizzi sono radicalmente differenti. Infatti gli sviluppatori hanno bisogno di accedere all'interfaccia del controllo per poterla utilizzare, mentre ciò non accade per gli utenti finali, i quali non hanno bisogno di questo tipo di accesso (e non dovrebbero solitamente averlo). Ciò che serve all'utente finale è che il controllo svolga il suo ruolo all'interno dell'applicazione.
Questo aspetto deve essere gestito a livello commerciale. Normalmente, uno sviluppatore di controlli distribuisce il controllo ad altri sviluppatori che lo utilizzano in ambiente di progettazione per creare le proprie applicazioni. Tali sviluppatori distribuiranno poi le proprie applicazioni al resto del mondo. Solitamente chi ha sviluppato i controlli probabilmente non desidera che tutti gli sviluppatori di applicazioni del mondo possano accedere all'interfaccia del controllo in modalità progettazione (se non pagando per ottenere tale accesso).
Visual Basic fornisce quindi un meccanismo che permette al controllo di determinare se si trova in modalità esecuzione o in modalità progettazione, e tale meccani-

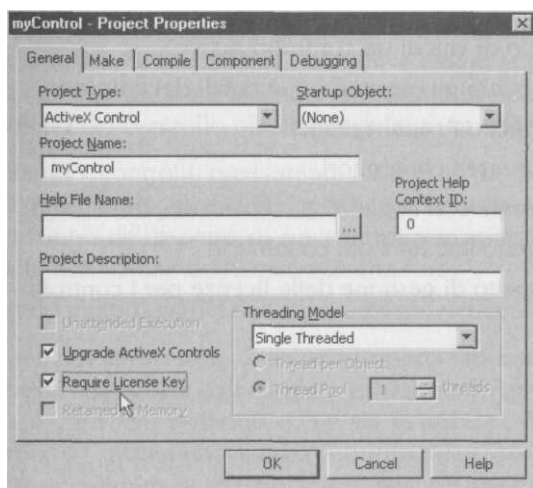
smo può essere utile per creare controlli che possano essere utilizzati liberamente dagli utenti finali in fase di esecuzione, ma che richiedano un file di licenza per esere utilizzati in modalità progettazione. Tutti questi aspetti di distribuzione dei controlli verranno approfonditi nel Capitolo 27.

Necessità di una licenza per lo sviluppatore

Una volta che avrete creato un controllo in grado di svolgere funzioni di una certa importanza e complessità (qualcosa per cui uno sviluppatore sia disposto a pagare) sarete sicuramente interessati all'implementazione di uno schema di gestione delle licenze che imponga agli sviluppatori di richiedere una licenza per poterlo utilizzare.

Questo aspetto nasconde numerose complessità, in quanto gli sviluppatori che utilizzeranno il vostro controllo potrebbero, a loro volta, includerlo all'interno di un controllo da loro sviluppato. Se aveste implementato la richiesta di una licenza per l'utilizzo in modalità progettazione, e tali sviluppatori avessero a loro volta introdotto un simile meccanismo, si renderebbe necessario, per chi utilizzasse questo nuovo controllo, possedere entrambe le licenze sulla propria stazione di lavoro; purtroppo, nonostante sia scomodo, questo è attualmente l'unico modo di procedere in questi casi.

Per definire una chiave di licenza per l'utilizzo del controllo ActiveX in modalità progettazione, è sufficiente selezionare la casella *Require License Key* nella scheda *General* della finestra di dialogo *Project Properties* prima di compilare il file .Ocx (come mostrato nella Figura 24.17).

Figura 24.17
Impostazione per la richiesta di una licenza.

Una volta compilato il controllo, avviate *VB Application Setup Wizard.* Il programma di setup risultante, quando eseguito, trasferirà la chiave di licenza al Registro di configurazione di un computer differente, permettendo l'utilizzo del controllo in ambiente di sviluppo. Setup Wizard è descritto in maggiore dettaglio nel Capitolo 35.

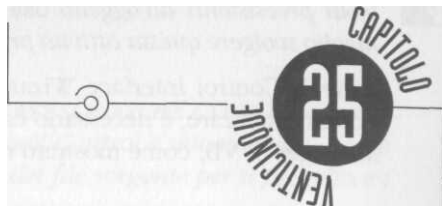
Copiando semplicemente il file .Ocx su un computer differente e registrandolo, la chiave della licenza non verrà trasferita, e senza di essa il controllo funzionerà solamente in modalità esecuzione, e non potrà essere utilizzato dagli sviluppatori in modalità progettazione! Infatti, se lo sviluppatore avesse una copia del controllo ma non possedesse la chiave nel Registro di configurazione, il controllo non sarebbe in grado di creare una propria istanza nell'ambiente di sviluppo.

Visual Basic crea un file con estensione . Vbl che contiene la chiave del Registro che fornisce la licenza per il controllo. Quando si utilizza Setup Wizard per creare una routine di installazione per il file .Ocx, il relativo file .Vbl viene automaticamente incluso nella routine.

Riepilogo

La possibilità di creare controlli ActiveX dotati di tutte le possibilità offerte da questa tecnologia utilizzando la versione 6 è un importante passo avanti per gli sviluppatori Visual Basic. I controlli ActiveX sono componenti ActiveX specializzati, descritti nel dettaglio nella Parte V del libro. In questo capitolo, che segue direttamente tale parte del libro, sono stati descritti i concetti necessari per comprendere la natura e il ciclo di vita dei controlli ActiveX. In questo capitolo:

- Avete imparato a utilizzare di controlli ActiveX in Ambiente VB.
- Avete imparato a creare progetti ActiveX Control.
- Avete imparato a lavorare con gli oggetti UserControl.
- È stata discussa la creazione di pacchetti di distribuzione contenenti controlli ActiveX.
- È stato descritto il ciclo di vita di un controllo ActiveX.
- Avete imparato a utilizzare gli eventi fondamentali del controllo.
- È stato introdotto l'oggetto PropertyBag.
- Avete imparato a utilizzare i contenitori.
- Avete imparato a lavorare con l'oggetto Extender dei contenitori.
- È stato descritto l'oggetto Ambient dei contenitori.
- È stato affrontato l'aspetto di gestione delle licenze per i controlli.

L'INTERFACCIA DEL CONTROLLO

- ActiveX Control Interface Wizard
- Come rendere funzionale il controllo
- Property Page Wizard
- Aggiunta eli una finestra di dialogo About al controllo

L'interfaccia di un controllo è costituita dai suoi membri (proprietà, eventi e metodi), come già spiegato nel Capitolo 24. Solitamente, il processo di creazione di un controllo è suddiviso in tre fasi:

- Creazione dell'aspetto del controllo
- Definizione dell'interfaccia del controllo
- Implementazione della logica necessaria per l'interfaccia

L'aspetto di un controllo (ovvero il modo in cui viene visualizzato sia in fase di progettazione sia in fase di esecuzione) *non* deve essere confuso con l'interfaccia del controllo, la cui definizione è già stata presentata in maniera dettagliata nel Capitolo 24. L'aspetto del controllo più essere definito in due maniere:

1. Posizionando controlli preesistenti all'interno di un nuovo controllo e manipolando le proprietà dell'oggetto `UserControl` *e* dei controlli inseriti. La definizione dell'aspetto di un controllo in questa maniera assomiglia molto alla definizione dell'aspetto di una form.
2. Utilizzando metodi grafici nell'evento `Paint` del controllo.

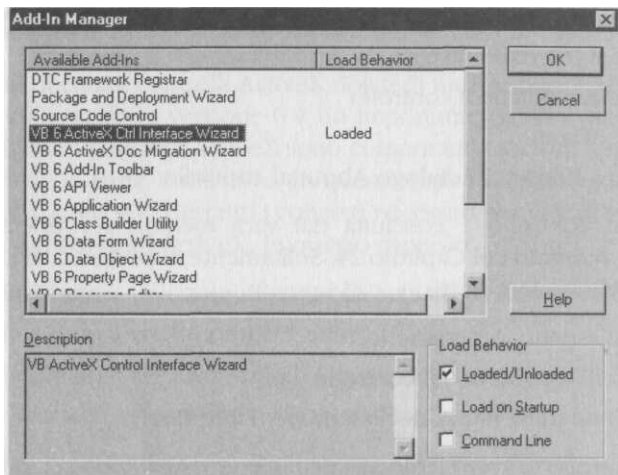
I controlli ActiveX creati con il secondo approccio vengono anche detti "user-drawn", ovvero "disegnati dall'utente", e verranno descritti nel Capitolo 26. In questo capitolo, invece, vedremo come sia possibile creare controlli seguendo il primo approccio, sfruttando *ActiveX Control Interface Wizard,* che semplifica enormemente il compito di creare l'interfaccia del controllo; nel primo paragrafo del capitolo vedremo come utilizzare questo strumento, mentre, in seguito, vedrerno come duplicare manualmente il codice di interfaccia creato dallo ActiveX Control Interface Wizard. Infine, vedremo come sia possibile aggiungere alcuni abbellimenti come, per esempio, le pagine delle proprietà e la finestra *About (Informazioni su)* al controllo.

ActiveX Control Interface Wizard

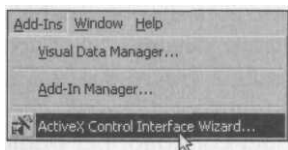
ActiveX Control Interface Wizard fornisce un ottimo punto di partenza per lo sviluppo dell'interfaccia di un controllo.

Se si desidera creare l'aspetto (non l'interfaccia) di un controllo aggiungendo con frolli preesistenti all'oggetto UserControl *e manipolandone le proprietà, è molto meglio svolgere questa attività prima di avviare Interface Wizard.*

ActiveX Control Interface Wizard è un'aggiunta di Visual Basic, per cui, prima di poterlo utilizzare, è necessario caricarlo (se già non è stato fatto) utilizzando *Add-in Manager &* VB, come mostrato nella Figura 25.1.

Figura 25.1 *Verìfica della presenza di ActiveX Control Interface Wizard in Add-in Manager.*

ActiveX Control Interface Wizard viene identificato da Add-in Manager come "VB 6 ActiveX Ctrl Interface Wizard".

Per aprire *Add-in Manager* è sufficiente scegliere la corrispondente voce del menu *Add-Ins.* Come mostrato in Figura 25.1, selezionate il Wizard e attivate la casella di opzione *Loaded/Unloaded* nel riquadro *Load Behaviour.* Una volta che il Wizard sarà stato caricato, apparirà una corrispondente voce nel menu *Add-Ins* (vedere Figura 25.2)

Figura 25.2 *Voce ActiveX Control Interface Wizard nel menu Add-Ins.*

Troverete maggiori informazioni su come utilizzare gli Add-in e Add-in Manager nel Capitolo 29.

per dimostrare l'utilizzo di ActiveX Control Interface Wizard creeremo un controllo ActiveX che abbia l'aspetto e si comporti come una casella di testo standard, aggiungendo alcuni membri personalizzati alla sua interfaccia.

Impostazione del controllo

Il file sorgente dell'oggetto UserControl *di* SelectText *si trova nel CD-ROM allegato sotto il nome di SelText.Ctl, mentre il progetto ActiveX Control è salvato con il nome SelText.Vbp. In Tabella 25.1 sono elencati i nomi dei file sorgente per il progetto e i moduli del controllo* SelectText.

Tabella 25.1 *Nomi dei file del controllo SelectText (SelText. Ocx).*

Nome del file	Contenuto
SelText.Ctl	Codice sorgente del modulo del controllo ActiveX
SelText.Vbp	Progetto del controllo ActiveX
Custom.Pag	Codice sorgente del modulo per la pagina delle proprietà personalizzata
FrmAbout.Frm	Finestra About del controllo
Testi.Frm	Form di test
Testi.Vbp	Progetto Standard EXE di test
Secrets.Bmp	Icona per la finestra Toolbox
SelText.Ocx	Controllo ActiveX compilato

L'idea sulla quale si basa il controllo SelectText non è niente di particolarmente elaborato, in quanto il controllo dovrà comportarsi esattamente come una normale casella di testo, eccetto per il fatto che, facendo clic su di esso, tutto il testo in esso contenuto dovrà essere selezionato.
Il controllo SelectText implementerà i membri personalizzati elencati nella Tabella 25.2, mentre nella Tabella 25.3 viene descritto come questi membri opereranno.

Tabella 25.2 *Membri personalizzati in SelText.Ctl.*

Nome del membro	Tipo	Tipo di dato	Valore predefinito
ClickEnabled	Proprietà	Boolean	False
SelectText	Metodo	N/D	N/D
onSelectText	Evento	N/D	N/D

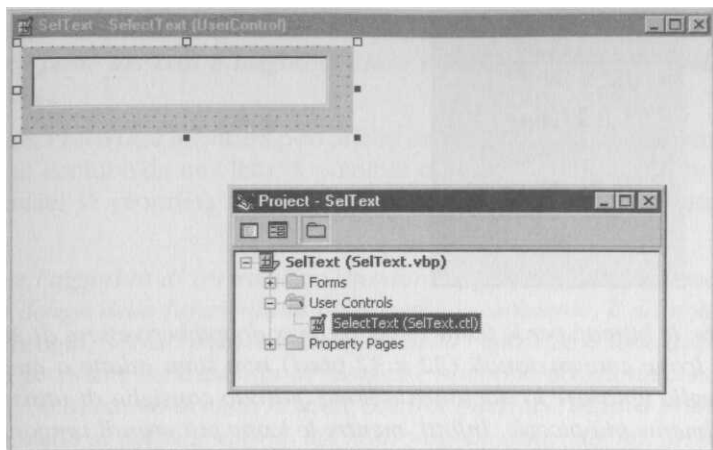
Tabella 25.3 *Funzionalità dei membri personalizzati di SelText.*

Nome del membro	Che cosa fa
ClickEnabled	Se ClickEnabled è impostato a True, tutto il testo contenuto nel controllo SelectText verrà selezionato quando il controllo riceverà l'evento clic
SelectText	Selezionerà tutto il testo contenuto nel controllo in seguito all'evento clic
onSelectText	Verrà innescato quando si verificherà una selezione in SelectText

È possibile disegnare la casella di testo (e gli eventuali altri controlli che costituiranno il nuovo controllo ActiveX) sia prima di avviare il Wizard sia una volta che il Wizard ha terminato il proprio compito. (In caso fosse necessario utilizzare di nuovo Interface Wizard per modificare il controllo, sarà possibile avviarlo nuovamente selezionando il controllo componente sul quale dovrà agire.)
È importante comprendere che Interface Wizard non si occupa di implementare i membri (se non attraverso il metodo della *delega,* grazie al quale un controllo eredita le proprietà, i metodi e gli eventi dei controlli che lo costituiscono). Questo significa che il programmatore deve comunque preoccuparsi di aggiungere il codice necessario a far funzionare i membri personalizzati. Infatti il codice aggiunto da Interface Wizard è essenzialmente uno scheletro, un modello di partenza.

Esiste una differenza tra gli eventi che il controllo può ricevere e quelli che invece genera. Il codice delle funzioni digestione degli eventi del controllo servono per fare in modo che il controllo possa rispondere agli eventi che riceve, mentre il codice di gestione scritto dagli sviluppatori che utilizzeranno il controllo serve a rispondere agli eventi generati dal controllo stesso.

Prima di tutto definite l'aspetto del controllo SelectText disegnando una normale casella di testo all'interno dell'oggetto UserControl. Utilizzate poi la finestra *Properties per* ripolire la proprietà relativa al testo predefinito, *default text(Text1),* ereditata dal controllo costituente (cioè quello che avete inserito nell'oggetto UserControl). Più avanti, nel corso del capitolo, nel paragrafo che spiega come rendere funzionale un controllo, vedremo come sia possibile aggiungere valori predefiniti per le proprietà Text e Caption di un controllo.
Assicuratevi che le dimensioni dell'oggetto UserControl di SelectText non siano eccessive. Nella Figura 25.3 viene mostrato un esempio delle dimensioni ottimali. Infatti le dimensioni predefinite del controllo ActiveX che stiamo creando, quando verrà aggiunto all'interno di un contenitore, saranno determinate dalla dimensione dell'oggetto UserControl, per cui è sempre meglio non esagerare.

Figura 25.3 *Ledimensioni dell'oggetto UserControl saranno quelte predefinite per il controllo ActiveX.*

Il controllo SelectText apparirà esattamente come una casella di testo standard (a ulteriore riprova del fatto che aspetto e interfaccia sono due cose differenti, per cui se l'aspetto è lo stesso, non è detto che l'interfaccia sia la stessa). Per fare in modo che la casella di testo costituente sia sempre della stessa dimensione dell'intero oggetto UserControl, aggiungete il codice mostrato nel Listato 25.1 all'evento Resize di UserControl.

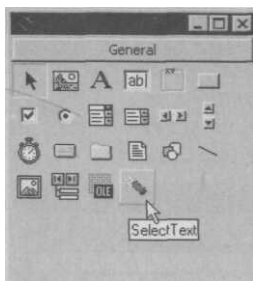
Listato 25.1 *Ridimensionamento del controllo costituenteper occupare tutta l'area client di UserControl.*

```
Private Sub UserControl_Resize()
   Text1.Move 0, 0, ScaleWidth, ScaleHeight
End Sub
```

L'evento Resize *di* UserControl *viene generato ogni volta che il controllo viene creato (o spostato). Vi accorgerete che la maggiorpane del codice riguardante l'inizializzazione e lafunzionalità del controllo viene inserito nella routine di gestione di questo evento.*

Aggiunta di un'icona Toolbox al controllo

È possibile aggiungere un'icona personalizzata per l'identificazione del controllo all'interno della Toolbox utilizzando la proprietà ToolboxBitmap dell'oggetto UserControl. Quando il controllo è selezionato, è possibile utilizzare la finestra di dialogo *Properties* per impostare questa proprietà (per il nostro esempio utilizzeremo una bitmap chiamata Secrets.Bmp mostrata nella Figura 25.4).

Figura 25.4 *Le dimensioni giuste per la bitmap personalizzate dalla casella sono 26x26 pixel.*

Idealmente, le bitmap per le icone della Toolbox dovrebbero essere di 26 x 26 pixel quindi le icone convenzionali (32 x 32 pixel) non sono adatte a questo scopo È inoltre meglio ignorare la documentazione quando consiglia di utilizzare bitmap sostanzialmente più piccole. Infatti, mentre le icone più grandi vengono irrimediabilmente distorte attraverso un'operazione di ridimensionamento, quelle troppo piccole si perdono nella Toolbox.

Esecuzione del Wizard

È venuto il momento di mettere in pratica tutte le nozioni apprese finora; avviate *Control Interface Wizard* scegliendo la relativa voce del menu *Add-Ins.* La prima finestra visualizzata dal Wizard ne riassume il compito. Se non desiderate rivedere in futuro la finestra, selezionate l'apposita casella di controllo al suo interno.
La finestra di dialogo successiva (mostrata in Figura 25.5) serve a selezionare i membri dell'interfaccia sui quali agire. Per default, il Wizard preseleziona tutti i membri standard del controllo (all'interno dell'elenco *Selected Names,* come mostrato in Figura 25.5)

Figura 25.5 *Finestra di Interface Wizard per la selezione dei membri dell'interfaccia.*

Anche se, attraverso questa finestra, sarebbe possibile rimuovere i membri standard, nella maggior parte dei casi è meglio lasciare invariate le selezioni nella casella Selected Names.

La finestra *Select Interface Members* può anche essere utilizzata per aggiungere altri membri di uso comune da una lista di possibili candidati. Per il nostro nuovo controllo selezionate le proprietà *SelLengtb, SelStart* e *SelText,* come mostrato nella Figura 25.5.

Generalmente l'aggiunta di un membro standard implica la sua implementazione attraverso la delega della funzionalità del controllo costituente. È da notare che le proprietà SelLength, SelStart *e* SelText *mostrate nella Figura 25.5 sono disponibili in quanto sono proprietà della casella di testo che abbiamo posizionato nell'oggetto* UserControl. *Se avessimo avviato ActiveX Control Interface Wizard prima di aver aggiunto la casella di testo all'interno del nostro nuovo controllo, tali proprietà non sarebbero apparse nella finestra* Select Interface Members.

La finestra successiva del Wizard permette di aggiungere membri di interfaccia personalizzati. Nella Figura 25.6 viene mostrata la finestra *Add Custom Member,* che appare facendo clic sul pulsante *New* nella finestra *Create Custom Interface Member.* La finestra *Add Custom Member* permette di dare un nome a proprietà, metodi ed eventi personalizzati.

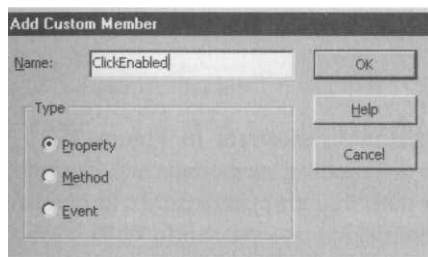
Figura 25.6 *Finestra Add Custom Member per la definizione di membri personalizzati.*

Potete utilizzare la lista *My Custom Members* della finestra *Create Custom Interface Members,* mostrata in Figura 25.7, per cancellare e modificare i nomi dei membri, oppure per crearne di nuovi. Per esempio, se il nome onSelectText non vi andasse bene per l'evento (supponiamo vogliate dare quel nome a un metodo, oppure che vogliate dare all'evento un nome più descrittivo) sarebbe possibile modificarlo utilizzando la finestra di dialogo *Edit Custom Member mostrata* nella Figura 25.8, accessibile facendo clic sul pulsate *Edit* della finestra *Create Custom Interface Members* .
La finestra successiva del Wizard, *SetMapping* (Figura 25.9) serve per definire una corrispondenza tra i membri del nuovo controllo e quelli dei controlli che lo costituiscono. Per esempio, se si crea un legame tra l'evento Change di UserControl e l'evento Change della casella di testo costituente, la casella di testo si prenderà carico di generare l'evento Change di UserControl.

Figura 25.7
Finestra Create Custom Interface Members per dare un nome ai membri personalizzati.

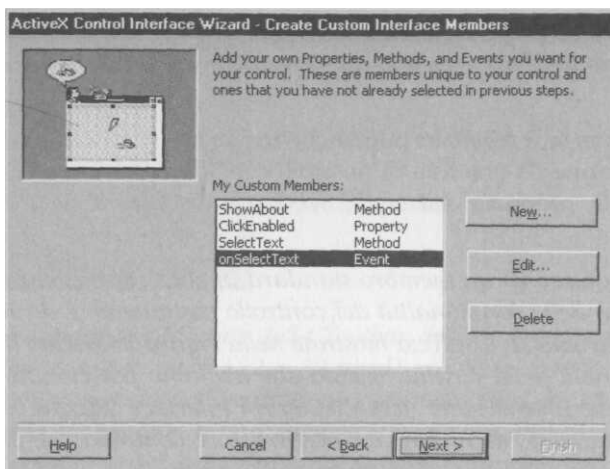
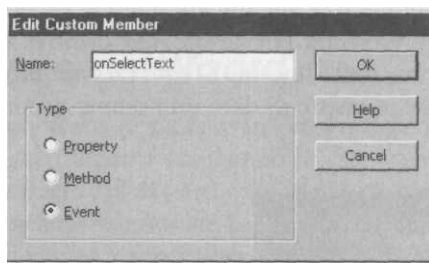



Figura 25.8
Finestra Edit Custom Member per la modifica dei nomi dei membri personalizzati.

A questo punto, la finestra *Set Attributes,* mostrata in Figura 25.10, permette di impostare gli attributi per ognuno dei membri personalizzati. Tramite questa finestra è possibile impostare il tipo di dati, il valore predefinito, la persistenza (ossia il fatto che il valore della proprietà venga salvato passando dalla modalità progettazione a quella di esecuzione e viceversa), il testo descrittivo che apparirà nella parte bassa della finestra *Properties* e infine gli argomenti, se applicabili.

Anche se i membri personalizzati non richiedono argomenti, i metodi e gli eventi hanno spesso questa necessità.

Per creare lo scheletro per il codice del controllo in base alle scelte effettuate attraverso il Wizard, fate ora clic su *Finish.* (Tenete presente che potrete avviare nuovamente il Wizard in qualsiasi momento per apportare cambiamenti al controllo utilizzando la sua interfaccia invece di agire direttamente dalla finestra *Code Editor,* cioè dall'editor del codice.)

Figura 25.9 *SetMapping permette di creare una corrispondenza tra eventi del nuovo controllo ed eventi dei controlli costituenti.*

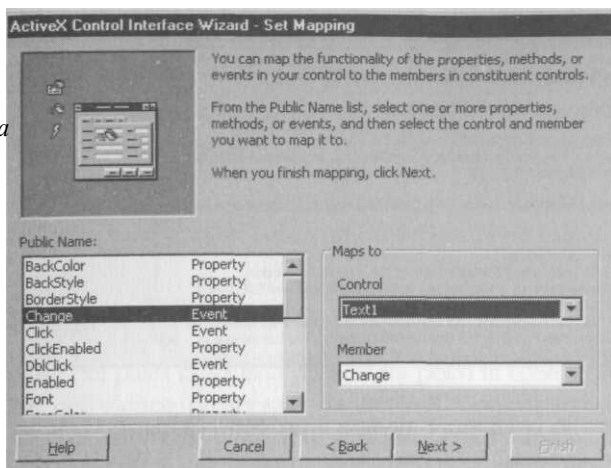
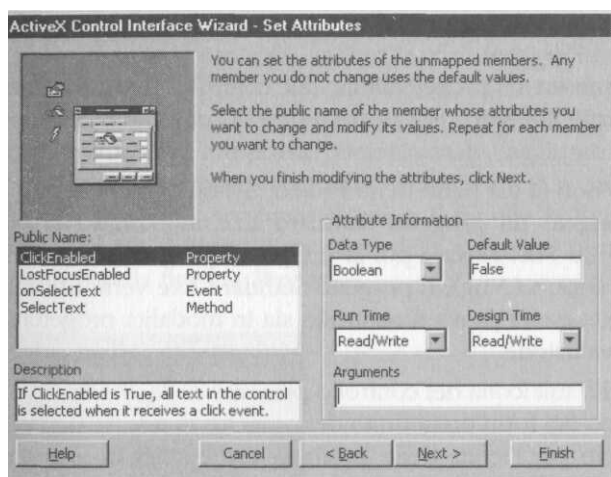
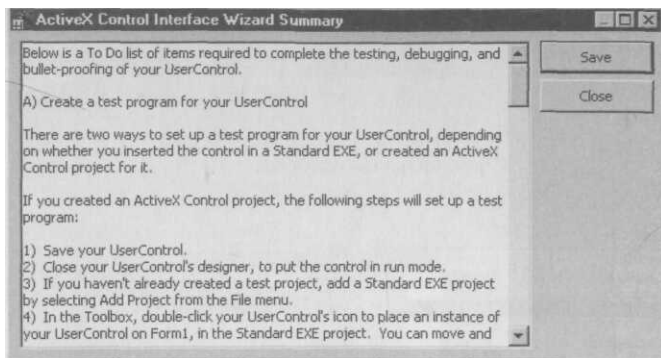



Figura 25.10 *SetAttributes permette di impostare informazioni sugli attributi dei membri personalizzati.*

Come ultimo favore verso il programmatore, Interface Wizard visualizza una lista di operazioni da fare *(To Do)* necessarie per effettuare il debugging dell'interfaccia del nuovo controllo, come mostrato nella Figura 25.11. Potete salvare e stampare l'elenco per potervi fare riferimento in seguito.
A questo punto è possibile procedere in due modi.
Per allontanare la possibilità di generare errori di "protezione generale" è solitamente meglio provare il controllo in un form piuttosto che in una pagina Internet Explorer, e compilare il controllo utilizzando la voce *Make* del menu *File.* Seguendo questo approccio per effettuare dei test (una volta compilato il controllo):

1- Aprite un progetto Standard Exe.
2. Utilizzate la finestra *Components* per aggiungere il nuovo controllo alla casella degli strumenti.

Figura 25.11 *Lista di cose da fare per effettuare il debugging del controtto.*

3. Inserite il controllo nel form e avviate il progetto di test.

La seconda possibilità consiste nel creare un Project Group, svolgendo le operazioni seguenti:

1. Salvate il controllo.
2. Chiudete la finestra di progettazione del controllo, ossia la finestra che serve a definirne l'aspetto. In questo modo il controllo entra in modalità esecuzione.
3. Se il controllo non fa già parte di un Project Group contenente un progetto di test, aggiungete un progetto Standard Exe utilizzando la voce *Add Project* del menu *File.* Salvate poi la coppia di progetti sotto forma di un file di Project Group (.Vbg). Il progetto Standard Exe verrà utilizzato d'ora in poi per mettere alla prova il controllo sia in modalità progettazione, sia in modalità esecuzione.
4. Fate doppio clic sull'icona del controllo presente nella *Toolbox* per aggiungerlo all'interno del form predefinito del progetto di test; se non avete definito una bitmap per l'icona della Toolbox, come spiegato in precedenza, verrà utilizzata un'icona di default.
5. Selezionate il nuovo controllo nel form e aprite la finestra *Properties;* assicuratevi che sia possibile consultare e modificare tutte le proprietà aggiunte al controllo.
6. Provate ora a modificare il valore di una delle proprietà personalizzate e a chiudere e riaprire il form. Assicuratevi quindi che le modifiche apportate alla proprietà siano state mantenute.
7. Aprite ora la finestra relativa al codice del controllo, e assicuratevi che tutti gli eventi personalizzati aggiunti al nuovo controllo appaiano nella casella di riepilogo a discesa di destra *(Procedure)* della finestra.
8. A questo punto, tornate alla finestra relativa al progetto di creazione del nuovo controllo (non alla form che serve per il suo test) e aggiungete o modificate il codice del controllo per aggiungervi funzionalità. È da notare che si renderà necessario eliminare l'istanza del controllo che avete aggiunto

nel form del progetto di test e aggiungervi una nuova istanza aggiornata, una volta terminate le modifiche al codice sorgente del nuovo controllo.

9. Aggiungete ora un po' di codice di esempio al controllo nel progetto di test, lavorando in modalità di progettazione (per esempio, aggiungete istruzioni MsgBox e Debug.Print a tutti gli eventi, per assicurarvi che funzionino correttamente). Invocate inoltre tutti i metodi per assicurarvi che vengano richiamati ed eseguiti correttamente. A questo punto, avviate il progetto di test e assicuratevi che tutto funzioni correttamente.

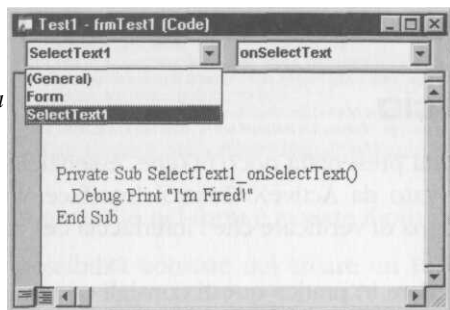
Verifica dell'interfaccia

La maggior parte dei passi della lista presentata poco fa (che, essenzialmente, erano elencati anche nell'elenco presentato da ActiveX Control Interface Wizard al termine della sua esecuzione) si occupa di verificare che l'interfaccia del controllo funzioni come ci si aspetta.
Ecco cosa si potrebbe fare per mettere in pratica questi consigli con il nostro nuovo controllo SelectText di esempio. Dopo aver chiuso la finestra di progettazione di UserControl, aggiungete il controllo SelectText al form di esempio del progetto di test. Aprite poi la finestra *Properties* e controllate che tutte le proprietà ereditarie selezionate tramite la finestra *Select Interface Members* del Wizard siano presenti. Verificate poi che anche le proprietà personalizzate, aggiunte tramite la finestra *Create Custom Interface Members,* siano disponibili e attive e che il loro tipo di dati e il loro valore di default siano quelli selezionati tramite la finestra *Set Attributes.* Nella Figura 25.12 è visualizzato il membro della proprietà personalizzata ClickEnabled all'interno della finestra *Properties.*

Figura 25.12 *Esempio di proprietà personalizzata nella finestra Properties.*

Dovete poi accertarvi che le modifiche alle proprietà personalizzate siano persistenti, caratteristica che è fondamentale per il buon funzionamento del controllo. Per effettuare questo controllo, modificate il valore di una proprietà personalizzata, per esempio modificando da False a True il valore di ClickEnabled; distruggete poi l'istanza del controllo SelectText (per esempio chiudendo il form che lo contiene); aprite quindi di nuovo il form per reinstanziare il controllo e assicuratevi che la proprietà ClickEnabled sia ancora impostata a True nella finestra *Properties.*

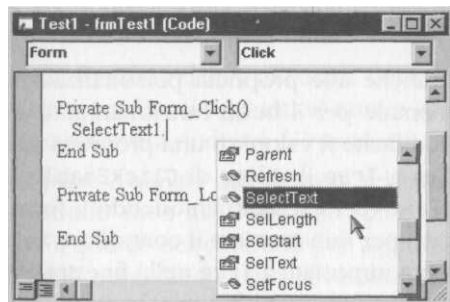
Assicuratevi poi che gli eventi del controllo personalizzato siano stati aggiunti alle infrastnitture di gestione degli eventi del form del progetto di test. Ricordate che gli eventi che sono membri del controllo vengono generati dal controllo e forniscono all'utente l'opportunità di rispondere adeguatamente al loro verificarsi. Tali eventi non vengono ricevuti dal controllo. La Figura 25.13 mostra la finestra *Code Editor* aperta con l'evento personalizzato onSelectText visualizzato.

Figura 25.13 *Esempio di verifica dell'infrastnittura di gestione degli eventi.*

Infine, ha senso verificare che i metodi personalizzati vengano invocati correttamente. Per effettuare questo controllo è necessario, prima di tutto, riaprire il modulo UserControl contenente il controllo e aggiungere poche istruzioni di codice alla procedura relativa al metodo:

```
Public Function SelectText() As Boolean
   Debug.Print "Invocato metodo di SelectText"
End Function
```

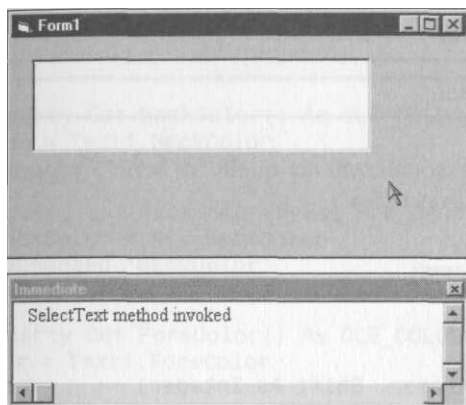
Se non avessimo aggiunto alcun codice per visualizzare il messaggio all'interno della funzione del metodo, non avremmo avuto modo di sapere se il metodo fosse stato realmente invocato.

A questo punto salvate il modulo UserControl, tornate al progetto di test, eliminate la vecchia istanza del controllo SelectText dal form e aggiungetene una nuova utilizzando la Toolbox. Adesso potete procedere aprendo la finestra *Code Editor* del form di test e aggiungendo del codice di test per verificare se il metodo del controllo viene invocato correttamente. La Figura 25.14 mostra come una invocazione di test sia stata aggiunta all'evento Click del form. È un buon segno che il metodo personalizzato di SelectText appaia nella casella di riepilogo a discesa *Properties/Methods*.

Figura 25.14 *Linea di codice per il test del metodo personalizzato.*

Ecco la semplice routine di gestione dell'evento clic che richiama il metodo personalizzato:

```
Private Sub Form_Click()
   SelectText1.SelectText
End Sub
```

Avviando il progetto e facendo clic sul form, apparirà la finestra *Immediate* che visualizzerà il messaggio indicante che il metodo è stato richiamato con successo (Figura 25.15).

Figura 25.15 *Messaggio di conferma dell'invocazione del metodo.*

Naturalmente è possibile utilizzare messaggi Debug.Print *per verificare completamente le funzionalità del controllo solamente se il controllo e il progetto di test fanno parte dello stesso Project Group in modalità di test. Se invece il controllo è stato compilato, per poterne mettere alla prova i metodi personalizzati, sarebbe necessario implementare qualche funzionalità, seppure limitata, per fare in modo che il risultato dell'esecuzione del metodo sia in qualche modo visualizzato all'interno del progetto di test.*

Bene! A questo punto tutti i test hanno dimostrato che ActiveX Control Interface Wizard ha svolto egregiamente il proprio lavoro e che l'interfaccia del controllo è completa. Naturalmente, il controllo non fa ancora nulla, ma presto ci occuperemo anche di questo aspetto.

Che cosa fa il Wizard?

Ricordate il Mago (Wizard) de "Il Mago di Oz"? Il grande e potente Oz. Naturalmente, al termine della storia, il mago si dimostra essere un comune mortale, al quale può capitare di fallire, e Dorothy si rende conto di poter fare a meno di lui. Per cui, una volta compreso il compito svolto da ActiveX Control Interface Wizard, vi potrebbe capitare di sentirvi come lei e di chiedervi "Posso fare a meno del Wizard?"

Interface Wizard si occupa di aggiungere codice al modulo UserControl, in base agli input che gli vengono forniti durante l'esecuzione. Osserviamo ora in dettaglio

il codice prodotto. Prima di tutto il Wizard aggiunge le definizioni delle costanti per i valori predefiniti di ogni proprietà personalizzata, come mostrato dal Listato 25 2

Listato 25.2 *Valori predefiniti delle proprietà.*

```
Const m_def_ClickEnabled = False
```

Poi, come è possibile notare nel Listato 25.3, vengono dichiarate le variabili per ogni proprietà personalizzata.

Listato 25.3 *Variabili delle proprietà.*

```
Dim m_ClickEnabled As Boolean
```

Tutti gli eventi (sia gli eventi personalizzati sia quelli "di serie") vengono dichiarati in seguito, come mostrato nel Listato 25.4.

Listato 25.4 *Dichiarazione degli eventi.*

```
Event Click()
Event DblClick()
Event KeyDown(KeyCode As Integer, Shift As Integer)
Event KeyPress(KeyAscii As Integer)
Event KeyUp(KeyCode As Integer, Shift As Integer)
Event MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)
Event MouseMove(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)
Event MouseUp(Button As Integer, Shift As Integer, X As Single, _
    Y As Single)
Event Change()
Event onSelectText()
```

Avrete notato come non ci sia alcuna differenza tra il modo in cui sono dichiarati gli eventi personalizzati e quello in cui sono dichiarati gli eventi di serie. Ogni proprietà membro dell'interfaccia del controllo viene dotata di due corrispondenti procedure Property Get e Property Let. Se la proprietà di un controllo costituente è stata messa in corrispondenza (in gergo, mappata) con una proprietà del nuovo controllo, il valore della proprietà del controllo costituente verrà assegnato a UserControl nella procedura Property Get e il valore della proprietà di UserControl verrà assegnato a quella del controllo costituente nella procedura Property Let.

Se al posto di una variabile venisse impostato un oggetto (come, per esempio, un oggetto font), la procedura Property Get *verrebbe accoppiata a una procedura* Property *Set (invece che Property Let).*

Il metodo PropertyChanged viene invocato nelle procedure Property Let. In questo modo il contenitore riceverebbe una notifica riguardo il fatto che il valore della proprietà è cambiato, e, di conseguenza, il controllo potrebbe svolgere le operazioni appropriate (per esempio la sincronizzazione dei valori delle proprietà). Il Listato 25.5 mostra alcune implementazioni delle proprietà.

Il tipo OLE_COLOR, *utilizzato nelle procedure* BackColor *e* ForeColor, *è definito nella libreria OLE Automation. Bisogna utilizzare la finestra di dialogo* Project References *per includere tale libreria nel progetto, per evitare la generazione di errori in fase di compilazione.*

Listato 25.5 *Implementazioni delle proprietà.*

```
Public Property Get BackColor() As OLE_COLOR
   BackColor = Text1.BackColor
End Property

Public Property Let BackColor(ByVal New_BackColor As OLE_COLOR)
   Text1.BackColor = New_BackColor
   PropertyChanged "BackColor"
End Property

Public Property Get ForeColor() As OLE_COLOR
   ForeColor = Text1.ForeColor
End Property

Public Property Let ForeColor(ByVal New_ForeColor As OLE_COLOR)
   Text1.ForeColor = New_ForeColor
   PropertyChanged "ForeColor"
End Property

Public Property Get Enabled() As Boolean
   Enabled = Text1.Enabled
End Property

Public Property Let Enabled(ByVal New_Enabled As Boolean)
   Text1.Enabled = New_Enabled
   PropertyChanged "Enabled"
End Property

Public Property Get Font() As Font
   Set Font = Text1.Font
End Property

Public Property Set Font(ByVal New_Font As Font)
   Set Text1.Font = New_Font
   PropertyChanged "Font"
End Property

Public Property Get BorderStyle() As Integer
   BorderStyle = Text1.BorderStyle
End Property

Public Property Get SelText() As String
   SelText = Text1.SelText
End Property
```

```
Public Property Let SelText(ByVal New_SelText As String)
   Text1.SelText = New_SelText
   PropertyChanged "SelText"
End Property

Public Property Get SelStart() As Long
   SelStart = Text1.SelStart
End Property

Public Property Let SelStart(ByVal New_SelStart As Long)
   Text1.SelStart = New_SelStart
   PropertyChanged "SelStart"
End Property

Public Property Get SelLength() As Long
   SelLength = Text1.SelLength
End Property

Public Property Let SelLength(ByVal New_SelLength As Long)
   Text1.SelLength = New_SelLength
   PropertyChanged "SelLength"
End Property

Public Property Get Text() As String
   Text = Text1.Text
End Property

Public Property Let Text(ByVal New_Text As String)
   Text1.Text = New_Text
   PropertyChanged "Text"
End Property

'Seguono le proprietà personalizzate
Public Property Get ClickEnabled() As Boolean
   ClickEnabled = m_ClickEnabled
End Property

Public Property Let ClickEnabled(ByVal New_ClickEnabled
   As Boolean)
   m_ClickEnabled = New_ClickEnabled
   PropertyChanged "ClickEnabled"
End Property
```

Il Wizard crea inoltre del codice, mostrato nel Listato 25.6, per implementare la delega degli eventi.

Listato 25.6 *Eventi delegati.*

```
Private Sub Text1_Change()
   RaiseEvent Change
End Sub

Private Sub Text1_Click()
   RaiseEvent Click
End Sub

Private Sub UserControl_DblClick()
```

```
    RaiseEvent DblClick
End Sub
Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)
    RaiseEvent KeyDown(KeyCode, Shift)
End Sub
Private Sub Text1_KeyPress(KeyAscii As Integer)
    RaiseEvent KeyPress(KeyAscii)
End Sub
Private Sub Text1_KeyUp(KeyCode As Integer, Shift As Integer)
    RaiseEvent KeyUp(KeyCode, Shift)
End Sub
Private Sub Text1_MouseDown(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    RaiseEvent MouseDown(Button, Shift, X, Y)
End Sub
Private Sub Text1_MouseMove(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    RaiseEvent MouseMove(Button, Shift, X, Y)
End Sub
Private Sub Text1_MouseUp(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    RaiseEvent MouseUp(Button, Shift, X, Y)
End Sub
```

Poi le proprietà personalizzate del controllo vengono inizializzate con i loro valori predefiniti, come mostrato dal Listato 25.7.

Listato 25.7 *Inizializzazione delle proprietà,*

```
Private Sub UserControl_InitProperties()
    m_ClickEnabled = m_def_ClickEnabled
End Sub
```

Come si vede nel Listato 25.8, viene poi aggiunta una routine che legge i valori delle proprietà persistenti dalla memoria, con PropertyBag.

Listato 25.8 *Caricamento dei valori delle proprietà dalla PropertyBag.*

```
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    Text1.BackColor=PropBag.ReadProperty("BackColor",_
        &H80000005)
    Text1.ForeColor=PropBag.ReadProperty("ForeColor",_
        &H80000008)
    Text1.Enabled = PropBag.ReadProperty("Enabled", True)
    Set Font = PropBag.ReadProperty("Font")
    Text1.SelText = PropBag.ReadProperty("SelText", "")
    Text1.SelStart = PropBag.ReadProperty("SelStart", 0)
    Text1.SelLength = PropBag.ReadProperty("SelLength", 0)
```

```
    Text1.Text = PropBag.ReadProperty("Text", "")
    m_ClickEnabled=PropBag.ReadProperty("ClickEnabled",_
        m_def JDlickEnabled)
End Sub
```

I valori letti dovranno anche essere scritti. Il Wizard aggiunge quindi una routine che scrive i valori delle proprietà, utilizzando sempre la PropertyBag, come mostrato nel Listato 25.9.

Listato 25.9 *Scrittura dei valori delleproprietà nella PropertyBag.*

```
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    Call PropBag.WriteProperty("BackColor", Text1.BackColor, _
        &H80000005)
    Call PropBag.WriteProperty("ForeColor", Text1.ForeColor,_
        &H80000008)
    Call PropBag.WriteProperty("Enabled", Text1.Enabled, True)
    Call PropBag.WriteProperty("Font", Font)
    Call PropBag.WriteProperty("SelText", Text1.SelText, "")
    Call PropBag.WriteProperty("SelStart", Text1.SelStart, 0)
    Call PropBag.WriteProperty("SelLength", Text1.SelLength, 0)
    Call PropBag.WriteProperty("Text", Text1.Text, "")
    Call PropBag.WriteProperty("ClickEnabled", m_ClickEnabled, _
        m_def_ClickEnabled)
End Sub
```

Questo è assolutamente tutto ciò che serve! È importante comprendere il codice generato dal Wizard perché, anche se lo si utilizza per avviare un progetto di un controllo personalizzato, è spesso molto più semplice modificare il codice a mano piuttosto che riawiare il Wizard per modificare l'interfaccia e, con la pratica, ve ne convincerete sempre di più.

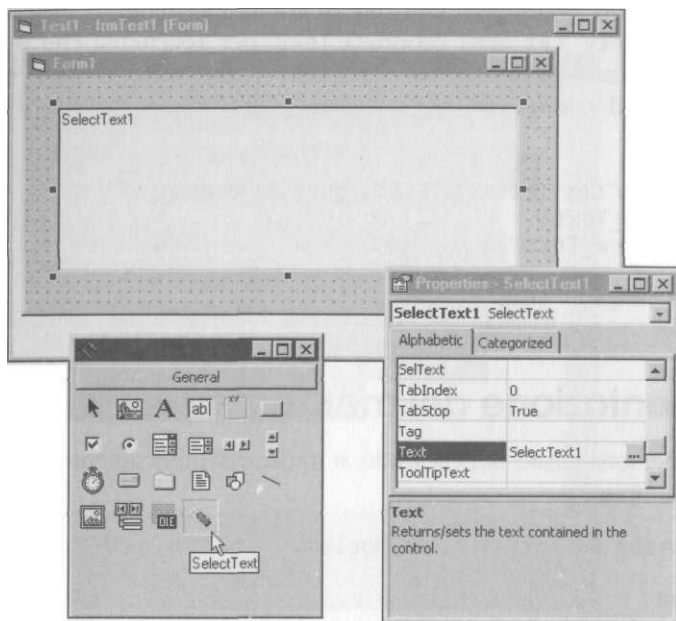
Come rendere funzionale il controllo

Rendere funzionale il controllo significa implementarne l'interfaccia; nel caso del nostro esempio, ciò significa aggiungere codice al controllo SelectText per aggiungere funzionalità alle due proprietà personalizzate del controllo stesso, ovvero il suo metodo personalizzato e il suo evento personalizzato. Ma ora occupiamoci di impostare il controllo SelectText per fare in modo che visualizzi la proprietà Default Text, ovvero il suo testo predefinito.

Aggiunta di un valore di testo predefinito

Aggiungendo una semplice casella di testo a un form, il controllo viene inizializzato con una proprietà Default Text del tipo "Textl", ossia il nome del controllo. E semplice, per l'oggetto Extender, aggiungere un valore predefinito per il testo o per la didascalia, (per una descrizione dettagliata dell'oggetto Extender, consultate il

paragrafo del Capitolo 24 che parla del rapporto tra controlli e contenitori.) Il nostro controllo di esempio si chiama SelectText. Il nome predefinito per la prima istanza del controllo è SelectText1. Quindi "SelectText1" è la stringa che deve apparire come valore predefinito del controllo SelectText1, come mostrato nella Figura 25.16.

Figura 25.16 *Utilizzo delle proprietà dell'oggetto Extender per impostare le proprietà del controllo.*

Per implementare questa funzionalità è necessario, prima di tutto, aggiungere una variabile, destinata a contenere la proprietà "text" del controllo. Se il Wizard non fosse stata aggiunta anche una proprietà Text delegata utilizzando, bisognerebbe anche aggiungere manualmente appropriate procedure Property.
A questo punto si renderà necessario aggiungere codice all'evento UserControl_-InitProperties di SelectText per fare in modo che la sua proprietà Text possa essere impostata al nome predefinito del controllo. Per questo particolare controllo, dato che il controllo casella di testo che lo costituisce è ben presente nel nuovo controllo, sarà necessario aggiungere del codice che modifichi anche Text1.Text. Senza questo intervento, il valore predefinito del testo verrebbe modificato, ma non verrebbe visualizzato nel controllo. Il Listato 25.10 mostra come modificare la proprietà del testo di default in modo che rispecchi il nome del controllo.

Listato 25.10 *Impostazione della proprietà testo predefinita al nome del controllo.*

```
'Variabili di proprietà:
Dim m_jext As String
'Inizializza le proprietà per lo UserControl
```

```
Private Sub UserControl_InitProperties()
   m_Text = Extender.Name
   Text1.Text = m_Text
End Sub
```

Sarà poi necessario aggiungere una riga di codice nel modulo UserControl All procedura Property Let della proprietà Text, per assicurarsi che il valore dell proprietà visualizzato nel controllo costituente venga sempre aggiornato quando l'utente modifica il valore della proprietà Text dell'istanza di SelectText nella finestra *Properties:*

```
Public Property Let Text(ByVal New_Text As String)
   m_Text = New_Text
   Text1.Text = m_Text 'Aggiunto per variazione dinamica
                       'della visualizzazione
   PropertyChanged "Text"
End Property
```

Implementazione del metodo SelectText

Un metodo personalizzato del controllo si traduce semplicemente in una funzione vuota. Per esempio:

```
Public Function SelectText() As Boolean

End Function
```

È quindi possibile aggiungere codice a piacere all'interno di tale funzione per implementare il metodo.

Dato che, per il nostro esempio, dobbiamo implementare la logica necessaria per selezionare il contenuto del controllo da diversi punti del codice, ha senso creare una nuova procedura per gestire l'operazione vera e propria, come mostrato nel Listato 25.11.

Listato 25.11 *Il metodo SelectText.*

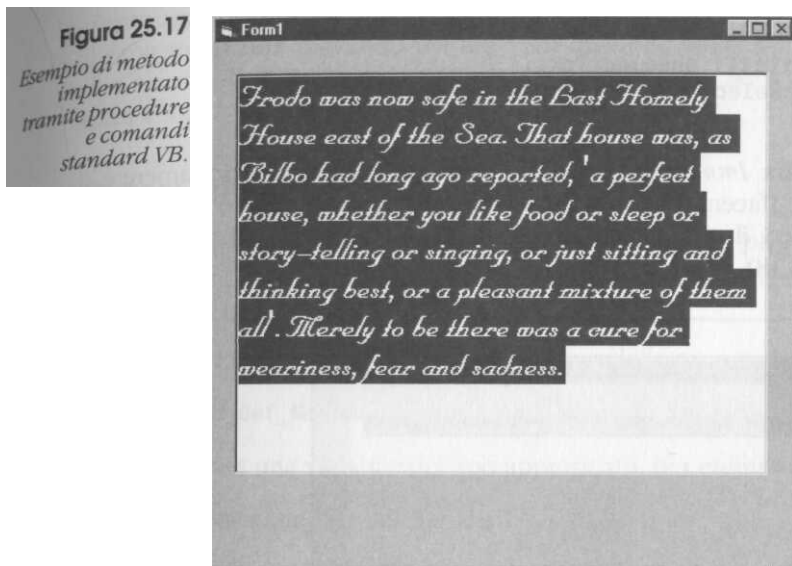
```
Public Function SelectText() As Boolean
   DoItToTheText
End Function

Private Sub DoItToTheText()
   Text1.SelStart = 0
   Text1.SelLength = Len(Text1.Text)
End Sub
```

La logica di questa procedura di selezione è già stata descritta nel paragrafo del Capitolo 16 riguardante la ricerca di file su disco.

Per verificare che il metodo funzioni, invochiamolo ora da un form di prova.

```
Private Sub Form_Click()
   SelectText1.SelectText
End Sub
```

Ora avviando il progetto di test, digitate del testo nel controllo SelectText, fate clic sul form e verificate che tutto il testo venga selezionato, come mostrato nella Figura 25.17.

Figura 25.17 *Esempio di metodo implementato tramite procedure e comandi standard VB.*

Il controllo SelectText *mostrato nella Figura 25.17 accetta più righe di testo. Questa modalità è stata attivata impostando a* True *la proprietà* Multiline *della casella di testo costituente.*

Implementazione dell'evento onSelectText

La responsabilità di assicurarsi che gli eventi personalizzati vengano innescati al momento giusto nei controlli personalizzati è lasciata al programmatore. Per svolgere questo compito, è possibile utilizzare l'istruzione RaiseEvent (naturalmente, l'evento deve essere dichiarato nel modulo UserControl). Dato che l'evento onSelectText deve essere innescato ogni volta che DoItToTheText viene invocata, ha senso inserire RaiseEvent all'interno di tale procedura. Ha inoltre senso verificare che la casella di testo contenga effettivamente del testo da selezionare, prima di innescare l'evento. Il Listato 25.12 presenta il codice revisionato.

Listato 25.12 *Innesco di un evento personalizzato.*

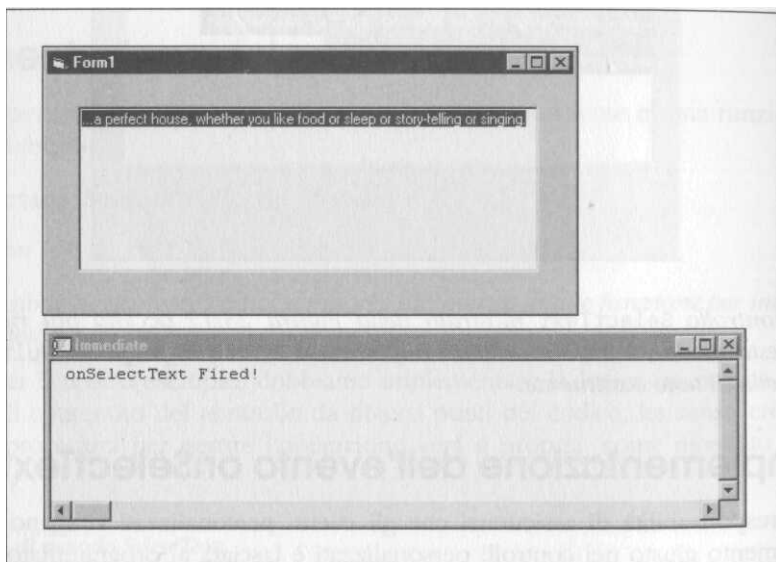
```
Private Sub DoItToTheText()
   If Text1.Text <> "" Then
      Text1.SelStart = 0
```

```
        Text1.SelLength = Len(Text1.Text)
        RaiseEvent onSelectText
    End If
End Sub
```

Per verificare che l'evento onSelectText venga effettivamente innescato, è necessario aggiungere codice all'interno della struttura di gestione degli eventi del progetto di test:

```
PrivateSubSelectText1_onSelectText()
    Debug.Print "onSelectText è stato generato!"
End Sub
```

Ora, aprite la finestra *Immediate* e avviate il progetto. Quando richiamerete il metodo SelectText (facendo clic sul form) il testo verrà selezionato e la finestra *Immediate* visualizzerà il messaggio indicante l'effettivo innesco dell'evento onSelectText (Figura 25.18).

Figura 25.18 *Utilizzo del comando RaiseEvent per innescare un evento.*

Implementazione delle proprietà personalizzate

La proprietà personalizzata ClickEnabled è già stata implementata (nel senso che il suo valore booleano è persistente tra la distruzione e la creazione delle istanze del controllo, e che la proprietà appare nella finestra *Properties).* Ciò che dovremo fare ora sarà dare un significato funzionale alla proprietà, facendo in modo che gli eventi di clic del controllo causino la chiamata alla funzione DoItToTheText solamente se la proprietà ClickEnabled avrà valore True. Ecco il codice della routine di gestione Text1_Click dell'oggetto UserControl che si occupa di raggiungere lo scopo:

```
Private Sub Text1_Click()
    If ClickEnabled Then
      DoItToTheText
    End If
    RaiseEvent Click
End Sub
```

Ovviamente è piuttosto semplice mettere alla prova questa funzionalità semplicemente aggiungendo la versione revisionata del controllo al form di prova, avviando il progetto di test, digitando del testo nel controllo SelectText, e facendo clic al suo interno. Il testo deve essere selezionato solamente se la proprietà ClickEnabled è impostata a True.

Riassunto

In sintesi, per rendere visibile (per esempio dalla finestra *Properties)* e funzionante una proprietà di un controllo è necessario aggiungere le seguenti porzioni di codice al modulo relativo a UserControl:

1. Definire una costante per il suo valore predefinito, per esempio:

   ```
   Const m_def_MessageText = "Vuoi proprio fare clic su di me?"
   ```

2. Dichiarare una variabile del tipo appropriato, per esempio:

   ```
   Dim m_MessageText As String
   ```

3. Scrivere le procedure Property Get e Property Let per la proprietà.
4. Aggiungere i metodi ReadProperties e WriteProperties alla PropertyBag sfruttando i metodi dell'oggetto UserControl, come mostrato poco fa.

Mi sembra che tutto questo non dovrebbe risultare molto più difficile di quanto non lo sia utilizzare il Wizard. Infatti, anche se ActiveX Control Interface Wizard gestisce egregiamente i dettagli della creazione delle proprietà personalizzate, non si occupa di farlo per i metodi e gli eventi. Un metodo dell'oggetto UserControl non è altro che una funzione pubblica. Il Wizard si limita ad aggiungere lo scheletro per il codice:

```
Public Function ShowMessage() As Variant

End Function
```

Il compito di aggiungere il codice che rende funzionante il metodo è comunque lasciato al programmatore (il tipo del valore di ritorno restituito dalla funzione dipende dalla scelta effettuata nella finestra di dialogo *SetAttributes).*
Per quanto riguarda gli eventi, Interface Wizard aggiunge le dichiarazioni degli eventi inclusi attraverso le finestre *Select Interface Members* e *Create Custom Interface Members :*

```
'Dichiarazioni di evento:
```

```
Event MouseDown(Button As Integer, Shift As Integer, _
   x As Single, Y As Single)

Event onShowMessage()
```

Questo è lo scheletro dell'infrastnittura necessaria per la gestione degli eventi quando il controllo viene inserito in un contenitore (per esempio un form). I parametri che vengono visualizzati nella finestra *Code Editar* per la routine di gestione dell'evento dipendono dagli argomenti elencati nella dichiarazione, stabiliti attraverso la finestra *Set Attributes* del Wizard.
Il compito di implementare il codice del modulo di UserControl che si occupa di generare l'evento è lasciato al programmatore. Quando questo accade, viene avviata l'esecuzione del codice che lo sviluppatore che utilizza il controllo ha inserito nella relativa routine di gestione dell'evento. Per generare un evento è sufficiente utilizzare il metodo RaiseEvent dell'oggetto UserControl:

```
RaiseEvent onShowMessage
```

Utilizzare il Wizard oppure non utilizzarlo? A voi la scelta. In ogni caso, sarà comunque necessario avere una buona familiarità con il codice di implementazione dell'oggetto UserControl.

Property Page Wizard

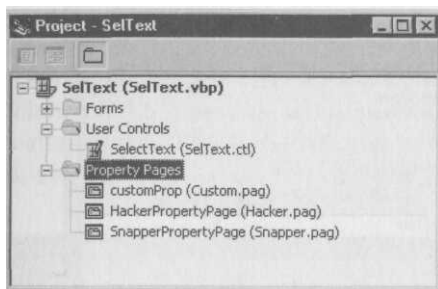
Le pagine delle proprietà *(property page)* sono un'interfaccia alternativa che può essere messa a disposizione degli sviluppatori che utilizzano il controllo, e sono accessibili tramite un pulsante nel campo relativo al valore della proprietà personalizzata "(Custom)" che appare in testa alla lista delle proprietà della finestra *Properties,* come mostrato nella Figura 25.19.

Figura 25.19 *Accesso alla pagina delle proprietà tramite il pulsante "..." della proprietà (Custom).*

Qualunque sviluppatore abbia utilizzato controlli ActiveX ha sicuramente familiarità con le pagine delle proprietà, che appaiono come finestre di dialogo suddivise in schede; ognuna delle schede permette all'utente del controllo di impostarne le proprietà attraverso controlli convenzionali (anziché attraverso la più scomoda interfaccia della finestra *Properties).*
Il codice sorgente di ogni pagina delle pagine delle proprietà associate a un controllo è memorizzato in un file .Pag, un file di testo ASCII simile, nella struttura, a un normale file di form di Visual Basic (.Frm). Il file contiene riferimenti interni a controlli incapsulati e alle relative proprietà e, in teoria, può essere modificato utilizzando un normale programma di editing di file di testo.

per modificare le pagine delle proprietà si utilizzano gli stessi strumenti che servono a lavorare sui form: la Toolbox, la finestra Properties e la finestra Code Editor.

Ogni file di pagine delle proprietà diventa poi un nodo di una cartella *Property pages,* all'interno della struttura del progetto, come è evidenziato da *Project Explorer* (Figura 25.20).

Figura 25.20 *Le pagine delle proprietà sono memorizzate in file e cartelle gestite da Project Explorer.*

E possibile aggiungere pagine delle proprietà al controllo in due modi:

- Avviando Property Page Wizard
- Scegliendo la voce *Add Property Page* dal menu *Project* mentre il controllo è selezionato.

Esecuzione di Property Page Wizard

Prima di poter avviare Property Page Wizard, è necessario abilitarlo utilizzando *Add-In Manager.* Poi, prima di avviare Property Page Wizard, assicuratevi di selezionare l'oggetto UserControl. La prima finestra del Wizard permette di aggiungere a piacimento le Property Page desiderate; il Wizard genererà poi tutte le Property Page per ogni pagina selezionata nella finestra di dialogo *Select Property Pages* (Figura 25.21). Le pagine StandardColor e StandardFont vengono create di default; se non le desiderate, assicuratevi di deselezionarle.
Il passo successivo consiste nell'aggiungere le proprietà alle Property Page, come Rostrato nella Figura 25.22.

Figura 25.21

La finestra Select Property Pages per la scelta delle Property Page.

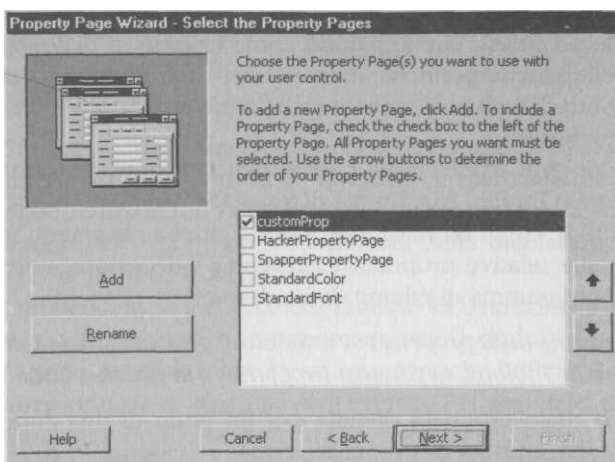
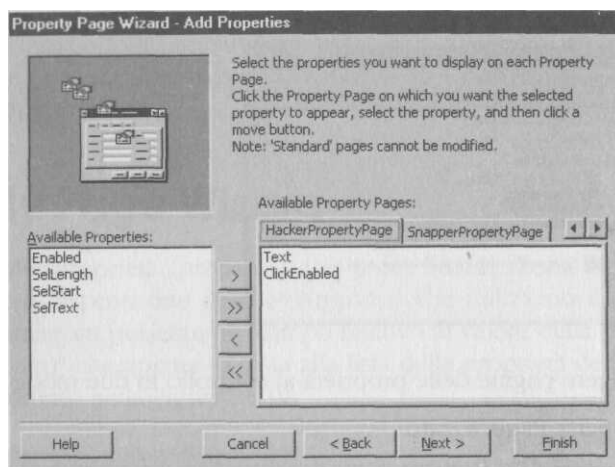



Figura 25.22

La finestra Add Properties per assegnare variabili alle proprietà.

In base alle informazioni fornite, il Wizard genererà una o più Property Page. Come passo finale viene presentato un elenco di operazioni utili per rendere funzionali le Property Page generate (solitamente, una serie di interventi nei punti in cui il Wizard ha inserito commenti "TO DO", contenenti le indicazioni su come operare). (Per Property Page particolarmente semplici, potrebbe anche non essere necessario svolgere alcuna operazione supplementare.)

Spesso vi capiterà di voler modificare l'aspetto delle Property Page generate dal Wizard. Per esempio, la Property Page di Figura 25.23 ha un casella di testo che originariamente ha la didascalia "Text.". Una didascalia più descrittiva sicuramente risulterebbe anche più utile. Per modificare l'aspetto di una Property Page potete utilizzare la finestra di progettazione della Property Page (e gli strumenti consueti di Visual Basic come la finestra *Properties e* la *Toolbox),* proprio nello stesso modo in cui modificate i normali form di Visual Basic.

Figura 25.23

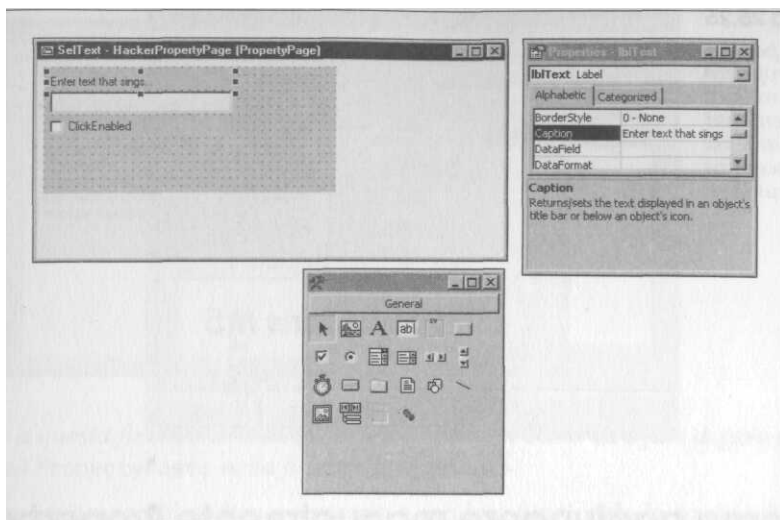
Per modificare i form e le Property Page si utilizzano gli stessi strumenti.

Chiudendo la pagina di progettazione della Property Page (e il controllo, se aperto) e aprendo il Form contenente il controllo, troverete un nuovo elemento "(Custom)" all'inizio dell'elenco delle proprietà nella finestra *Properties.* Facendo clic sul pulsante accanto alla casella di testo del relativo valore, verranno visualizzate le Property Page che avete creato finora, come mostrato dalla Figura 25.24.

Figura 25.24

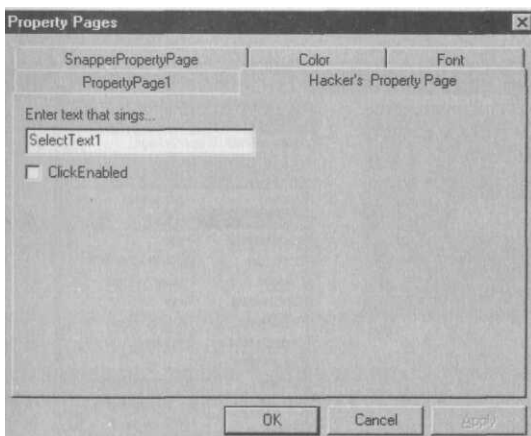
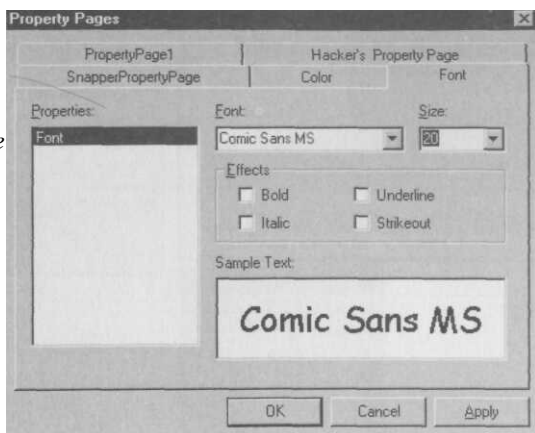
Le Property Page sono visualizzate in una finestra di dialogo a schede.

La pagina delle proprietà Standard Font, *generata automaticamente dal Wizard, è completamente funzionante e piuttosto versatile ed efficace (Figura 25.25).*

Figura 25.25
Lapagina delle proprietà Standard Font generata automaticamente dallaprocedura guidata.

Come aggiungere manualmente Property Page

Per aggiungere manualmente una Property Page, dovete innanzitutto aggiungere una Property Page al progetto utilizzando la voce *Add Property Page* del menu *Project.* Poi, dovete aggiungere al suo interno i controlli necessari e manipolarne le proprietà attraverso la finestra di progettazione della Property Page. Il processo è identico alla creazione e alla modifica di un form standard di Visual Basic. Nella Figura 25.26 è mostrata una Property Page personalizzata (salvata con il nome Custom.Pag) in modalità progettazione.

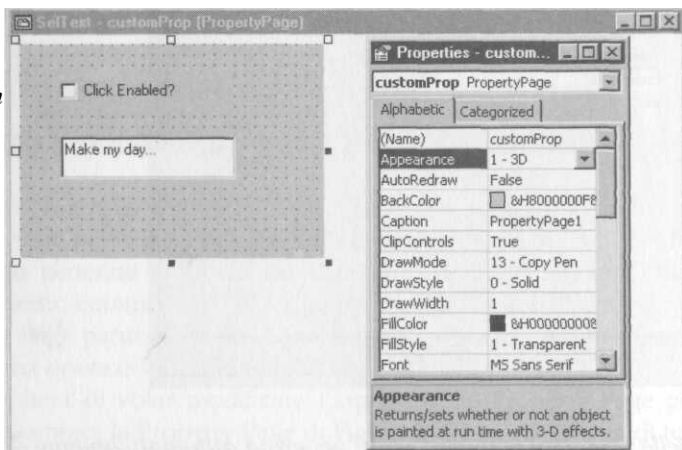
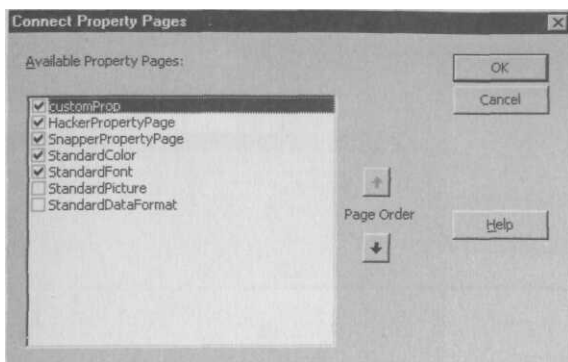
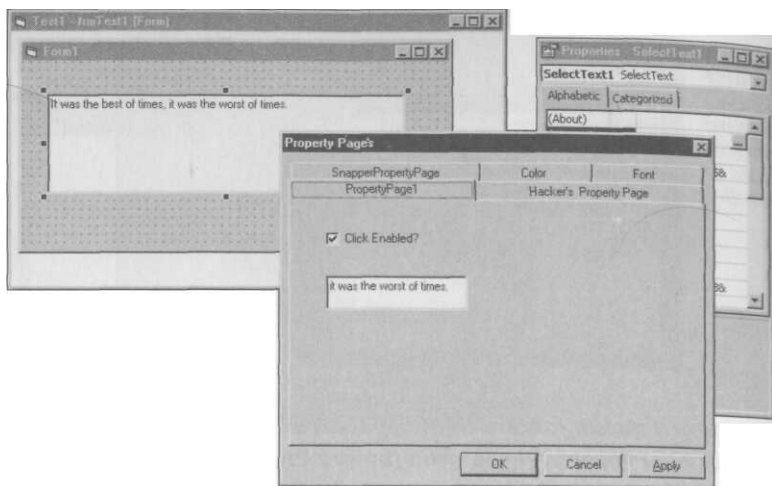
Figura 25.26
Una Property Page personalizzata vieneprogettatain manieraidentica a unform.

Il passo successivo consiste nel collegare la Property Page al controllo, utilizzando la finestra di dialogo *ConnectProperty Pages,* mostrata in Figura 25.27.

Figura 25.27 *La finestra Connect PropertyPages per collegare le proprietà personalizzate al controllo.*

Per accedere a questa finestra di dialogo, selezionate UserControl *e fate doppio clic sulla proprietà* PropertyPages *nella finestra* Properties.

Infine, dovete aggiungere il codice necessario a rendere funzionale la pagine delle proprietà. Tale codice non è necessariamente lungo e complesso (vedere Listato 25.13 per un esempio). Esso dovrà essere modificato a seconda dei nomi dei controlli contenuti nella pagina e in base ai nomi delle proprietà dell'oggetto UserControl alle quali dovranno essere collegati. Nella Figura 25.28 è mostrata la pagina delle proprietà personalizzata e le proprietà di UserControl che fanno riferimento al codice presentato nel Listato.

Listato 25.13 *Codice necessario per rendere funzionale la pagina delle proprietà personalizzata.*

```
Option Explicit
Private Sub chkClickEnabled_Click()
   Changed = True
End Sub

Private Sub txtText_Change()
    Changed = True
End Sub

Private Sub PropertyPage_ApplyChanges()
   SelectedControls(0).ClickEnabled = _
      (chkClickEnabled.Value = vbChecked)
   SelectedControls(0).Text = txtText.Text
End Sub

Private Sub PropertyPage_SelectionChanged()
   chkClickEnabled.Value = (SelectedControls(0).ClickEnabled _
      And vbChecked)
   txtText.Text = SelectedControls(0).Text
End Sub
```

Figura 25.28
Non è eccessivamente difficile rendere funzionale una Property Page.

Aggiunta di una finestra di dialogo About al controllo

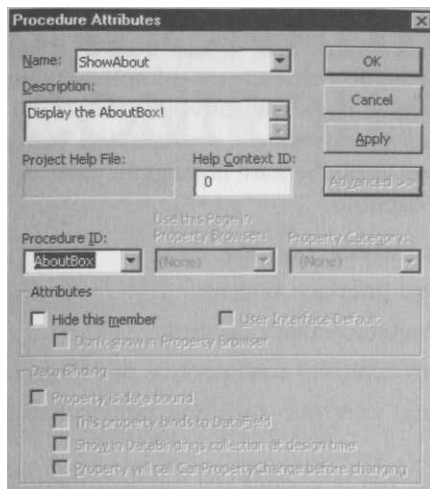
Aggiungere una finestra *About* di informazioni sul controllo è molto semplice. In questo modo, gli sviluppatori che utilizzeranno il controllo potranno visualizzarla in fase di progettazione facendo clic sul pulsante che si trova accanto alla proprietà *About* nella finestra *Properties.* È possibile utilizzare la finestra *About* per fornire informazioni riguardo la proprietà, l'autore, il copyright e altre notizie utili riguardanti il controllo.
Prima di tutto create la form *About* e aggiungetela al progetto ActiveX Control. (Nel nostro esempio, il form si chiamerà `dlgAbout`, lo stesso utilizzato nel Capitolo 19.)

Assicuratevi di aggiungere il form About *al progetto del controllo, non al progetto di test, che potrebbe far parte dello stesso Project Group, altrimenti il form non sarebbe accessibile dal controllo,*

Poi aprite la finestra *Code Editar per* l'oggetto `UserControl` e aggiungete la seguente procedura:

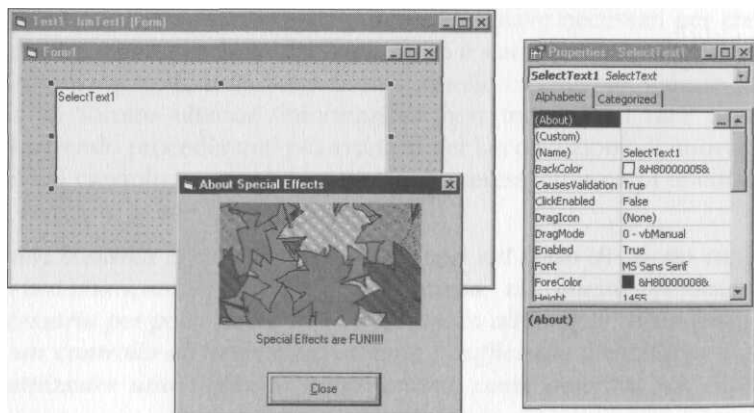
```
Public Sub ShowAbout()
   dlgAbout.Show vbModal
   Unload dlgAbout
   Set dlgAbout = Nothing
End Sub
```

Sempre con la finestra *Code Editor* aperta scegliete la voce *Procedure Attributes* de menu *Tools.* Fate poi clic sul pulsante *Advanced* per espandere la finestra di dialogo *Procedure Attributes,* come mostrato nella Figura 25.29.

Figura 25.29
Definizione di una procedura per la visualizzazione della finestra About.

Assicuratevi che *ShowAbout* sia selezionata nella lista *Name, e* assegnatele il *Procedure ID* di *AboutBox.* Se lo desiderate, è anche possibile aggiungere una descrizione del tipo "Visualizza la AboutBox!", che apparirà nella finestra *Properties* quando viene selezionata la finestra *About.*

È possibile assegnare la stessa finestra About *anche a più controlli dello stesso progetto contemporaneamente.*

Aprendo ora il form del progetto di test e selezionando il controllo SelectText, nella finestra *Properties* apparirà un campo *About* all'interno della lista delle proprietà. Facendo clic sul pulsante alla destra del campo *About,* verrà visualizzata la finestra *About,* come mostrato dalla Figura 25.30.

Figura 25.30
È semplice aggiungere un tocco di professionalità ai propri controlli.

Riepilogo

Questo capitolo ha descritto nel dettaglio come creare e implementare l'interfaccia di un controllo personalizzato semplice ma significativo: `SelectText`. Esso si basa su un controllo costituente (una casella di testo), sfruttandone l'interfaccia attraverso il meccanismo della delega per implementare la maggior parte delle funzionalità del nuovo controllo. In questo capitolo:

- Avete imparato a definire l'aspetto del controllo.
- È stato descritto ActiveX Control Interface Wizard.
- Avete imparato a creare un progetto ActiveX Control.
- È stato presentato il concetto di controllo costituente e di delegazione.
- Avete imparato ad assegnare al controllo un'icona personalizzata per la Toolbox.
- È stata descritta la metodologia per la verifica del funzionamento del controllo.
- Avete imparato a creare il codice per l'interfaccia del controllo.
- È stato spiegato come rendere funzionale il controllo.
- Avete imparato a definire valori predefiniti per le proprietà del controllo.
- È stato spiegato come implementare proprietà personalizzate.
- Avete imparato a utilizzare Property Page Wizard.
- È stato descritto il metodo da seguire per aggiungere manualmente al controllo delle pagine delle proprietà.
- È stato spiegato come associare una finestra About al controllo.

LE FUNZIONALITÀ DEL CONTROLLO

- Creazione del controllo StickyFrame
- Aggiunta di tipi enumerati all'interfaccia del controllo
- Aggiunta di proprietà enumerate personalizzate
- Impostazione di una proprietà predefinita
- Creazione di una proprietà predefinita per l'interfaccia utente
- Aggiunta di finestre di dialogo personalizzate alla finestra Properties
- Raggruppamento di proprietà per categorie
- Proprietà in fase di progettazione e in fase di esecuzione
- Creazione di proprietà valide solo in fase di esecuzione
- Creazione di un controllo basato su più controlli costituenti
- Controlli user-drawn
- Creazione di un controllo "Confetti"
- Gli oggetti UserControl

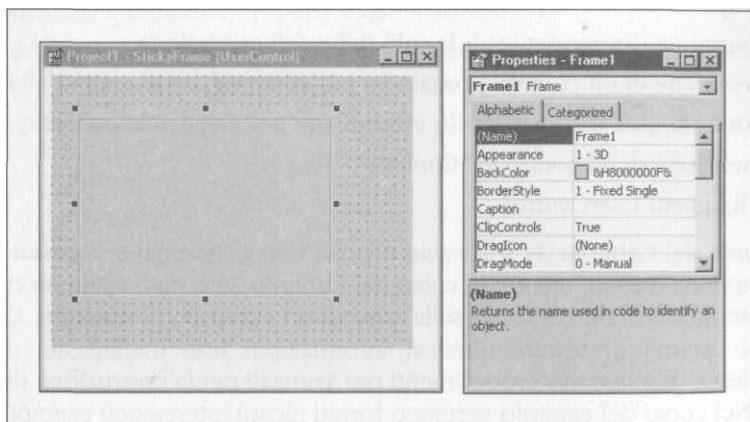
Nel corso del Capitolo 24 sono stati esposti i concetti chiave necessari per creare un proprio controllo, mentre le corso del Capitolo 25 è stato spiegato come costruire l'interfaccia di un controllo sulla base di un controllo costituente. Questo capitolo si occupa di fornire ulteriori informazioni non trattate nei due capitoli precedenti, descrivendo procedimenti più avanzati per la costruzione di nuovi controlli. Nel corso del capitolo verranno forniti alcuni interessanti esempi di controlli personalizzati.

Come per gli altri controlli creati finora, quelli creati nel corso di questo capitolo non verranno automaticamente registrati nel sistema, ma questa operazione è comunque necessaria per poter utilizzare un controllo all'interno di un progetto. Per registrare un controllo all'interno del sistema è sufficiente compilarne il progetto oppure utilizzare una utility di registrazione, come descritto più volte in questo libro.

Il controllo StickyFrame

Probabilmente vi ricorderete dell'esempio del Capitolo 11 che dimostrava come utilizzare le funzioni GetDesktopWindow, GetWindowRect e ClipCursor per restringere il raggio d'azione del cursore all'interno dell'area di uno specifico controllo. Nel corso del Capitolo 14 l'esempio è stato poi riciclato per dimostrare l'utilizzo di moduli di classe per creare un controllo Frame delegato. In altre parole, il controllo Frame predefinito è stato semplicemente "incartato" assieme al modulo di classe per creare un nuovo tipo di Frame dotato dei metodi Stick e UnStick. Le tecniche dimostrate nel Capitolo 14 non erano però dipendenti dalla capacità di creare controlli ActiveX.
Chiaramente la creazione di un controllo ActiveX completamente nuovo è una soluzione di gran lunga migliore, dato che, per utilizzarlo poi all'interno di un progetto, sarà sufficiente aggiungere il controllo dalla Toolbox. Sarà poi il meccanismo ActiveX a prendersi cura di creare e distruggere le istanze del controllo, il che significa (differentemente dalla dimostrazione del Capitolo 14) che lo sviluppatore non dovrà più preoccuparsi di creare metodi Create e Destroy per il controllo.

// codice sorgente per il controllo è salvato il un modulo chiamato Sticky.Ctl, appartenente a un Project Group chiamato Sticky. Vbp. Il primo passo consiste nell'aggiungere il controllo Frame *all'interno dell'oggetto* UserControl, *come mostrato nella Figura 26.1.*

Figura 26.1 *StickyFrame si basa sul controllo Frame, incluso nell'oggetto UserControl.*

Per assicurarsi che StickyFrame abbia le stesse dimensioni del controllo Frame costituente, aggiungete il codice appropriato all'evento Resize di UserControl:

```
Private Sub UserControl_Resize()
   Frame1.Move 0, 0, ScaleWidth, ScaleHeight
End Sub
```

Inoltre, dato che StickyFrame deve comportarsi come un normale Frame (dotato di due metodi aggiuntivi) la maggior parte dei membri del controllo Frame dovrà essere messa in corrispondenza con elementi nell'oggetto UserControl di StickyFrame.

È possibile utilizzare la lista di proprietà del controllo Frame, visualizzata della finestra Properties *(vedere Figura 26.1), per definire la lista delle proprietà del controllo constituente da mappare su quelle del nuovo controllo. Oltre alle proprietà mostrate dalla finestra* Properties, *dovete ricordarvi di mappare la proprietà* hWnd *del controllo* Frame *costituente. Infatti la funzione* GetWindowRect *ha bisogno di ricevere, come parametro, un handle a una finestra e di conseguenza il metodo* Stick *non funzionerebbe senza tale handle.*

Sta a voi decidere se creare il codice per la definizione dell'interfaccia di StickyFrame (e mappare le proprietà del controllo costituente) a mano oppure utilizzando ActiveX Control Interface Wizard. (Per una spiegazione di entrambi i metodi consultate il Capitolo 25.) In ogni caso, si renderà necessario definire la struttura interna del codice con routine Property e routine per il controllo della persistenza. I membri, per esempio, devono essere dichiarati come segue:

```
Event Click()
```

Bisogna poi aggiungere le procedure per le proprietà, con le opportune mappature dei costituenti:

```
Public Property Get BorderStyle() As Integer
   BorderStyle = Frame1.BorderStyle
End Property
Public Property Let BorderStyle(ByVal New_BorderStyle As Integer)
   Frame1.BorderStyle() = New_BorderStyle
   PropertyChanged "BorderStyle"
End Property

Public Property Get hWnd() As Long
   hWnd = Frame1.hWnd
End Property
```

Dato che la proprietà hWnd *del* Frame *(ossia l'handle atta finestra) è di sola lettura, bisogna implementare solamente una procedura* Property Get *(non accoppiata a una corrispondente procedura* Property Let *o Set).*

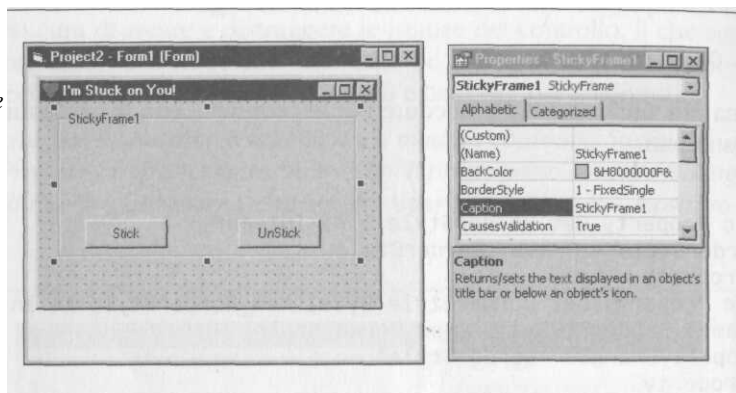
La persistenza delle proprietà e il ciclo di vita del controllo vengono gestiti dal codice che segue:

```
 'Carica valori di proprietà dalla memoria
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)

   Frame1.BorderStyle = PropBag.P»eadProperty( "BorderStyle", 1)

End Sub

'Scrive i valori di proprietà nella memoria
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)

   CallPropBag.WriteProperty("BorderStyle",_
```

```
        Frame1.BorderStyle,     1)
End Sub
```

Sarebbe poi utile che il nome dell'istanza del controllo StickyFrame venisse inizia lizzato automaticamente al valore della proprietà Caption (per esempio, Stickv Frame1, come mostrato nella Figura 26.2). Per ottenere questo risultato è sufficiente aggiungere il codice seguente all'evento InitProperties dell'oggetto UserControl, in modo da assegnare il valore di Extender.Name alla proprietà Caption del controllo costituente:

```
Private Sub UserControl_InitProperties()
    Frame1.Caption = Extender.Name
End Sub
```

Figura 26.2
I controlli ActiveX per default devono avere come valore iniziale della proprietà Caption il nome del controllo.

A questo punto è arrivato il momento di implementare i due metodi del controllo, Stick e UnStick. Prima di tutto aggiungete i tipi e le dichiarazioni per le API (APIDec.Bas) al progetto UserControl, e dichiarare un ambito locale per la variabile di UserControl di tipo RECT, che dovrà contenere le dimensioni del rettangolo all'interno del quale dovrà essere limitato il movimento del cursore. Dichiarate inoltre una variabile di tipo Boolean (chiamandola Stuck) che indicherà se il movimento del cursore sarà ristretto o meno all'interno del rettangolo. Tale variabile dovrà essere inizializzata a False in fase di creazione dell'istanza di UserControl (ovvero durante l'evento Initialize):

```
Private FrameRect As RECT
Private Stuck As Boolean
Private Sub UserControl_Initialize()
    Stuck = False
End Sub
```

L'implementazione dei metodi Stick e UnStick è piuttosto semplice:

```
Public Function Stick() As Boolean
    Stick = False
    GetWindowRect Me.hwnd, FrameRect
```

```
   ClipCursor FrameRect
   Stuck = True
   Stick = True
End Function

Public Function UnStick() As Boolean
   Dim ScreenRect As RECT, ScreenHandle As Long
   UnStick = False
   ScreenHandle = GetDesktopWindow
   GetWindowRect ScreenHandle, ScreenRect
   ClipCursor ScreenRect
   Stuck = False
   UnStick = True
End Function
```

È buona norma definire un valore di ritorno per i metodi, destinato a indicare se sono stati eseguiti con successo. Non sarà comunque obbligatorio, per gli utenti, controllarne il valore, ma sarà comunque utile offrirne la possibilità.

È inoltre consigliabile definire una routine che si occupi di rendere più sicuro il controllo, ovvero per garantire che il cursore non resti bloccato nell'area ristretta quando il controllo viene distrutto, anche se l'utente si è dimenticato di richiamare il metodo UnStick. Il codice per implementare questo accorgimento deve essere inserito nella routine di gestione dell'evento Terminate:

```
Private Sub UserControl_Terminate()
   If Stuck Then
      Me.UnStick
   End If
End Sub
```

Per verificare il funzionamento del controllo utilizzate la finestra *Project Properties* per dare al progetto il nome *StickeeFrame* (Figura 26.3) e compilate il controllo selezionando la voce *Make Sticky.Ocx* del menu *File.*

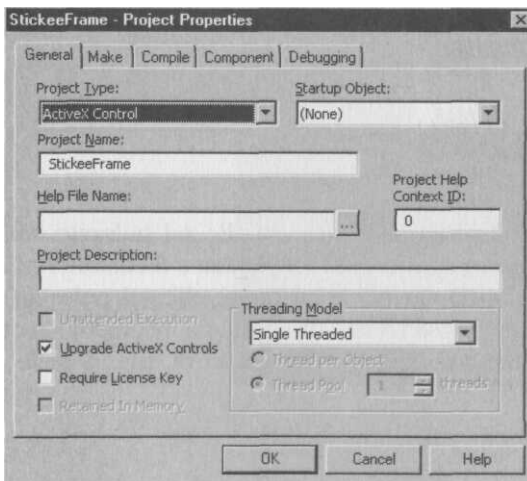
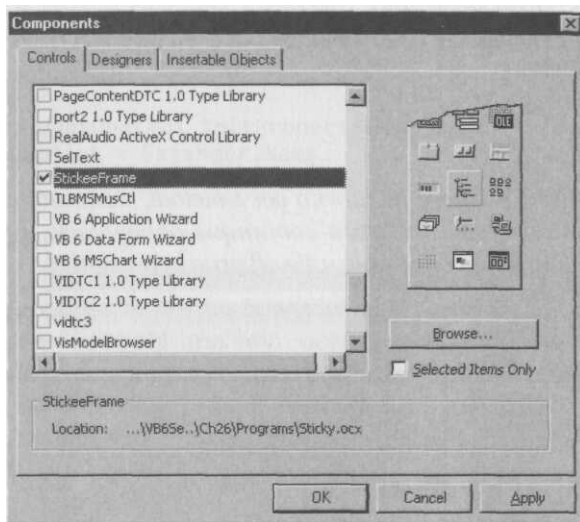
Figura 26.3
Il nome visualizzato nella lista Components sarà uguale a quello del progetto del controllo.

A questo punto aprite un 'altra istanza di VB6 e create un progetto di test (un progetto di esempio, TestStik. Vbp, si trova nel CD-ROM allegato). Utilizzate quindi la finestra di dialogo Components *per aggiungere il nuovo controllo alla Toolbox, come mostrato nella Figura 26.4. Infine utilizzate la Toolbox per aggiungere il nuovo controllo StickyFrame alform delprogetto di test.*

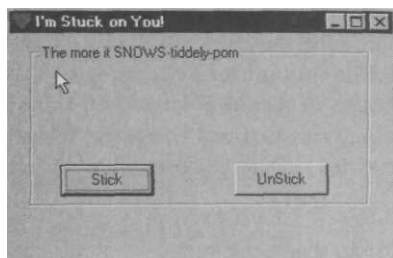
Figura 26.4
Aggiunta del nuovo controllo alla Toolbox tramitelafinestra Components.

Dato che, probabilmente, il controllo non sarà stato ancora registrato nel sistema, il progetto TestStick genererà errori di caricamento. Per risolvere ilproblema, assicuratevi di registrare correttamente il controllo nel sistema e poi eliminate e ricreate l'istanza del controllo all'interno del form.

Aggiungete poi due pulsanti, cmdStick e cmdUnstick, all'interno di StickyFrame. La chiamata ai due metodi del controllo StickyFrame è molto semplice:

```
Private Sub cmdStick_Click()
    StickyFrame1.Stick
End Sub
Private Sub cmdUnStick_Click()
     StickyFrame1.UnStick
End Sub
```

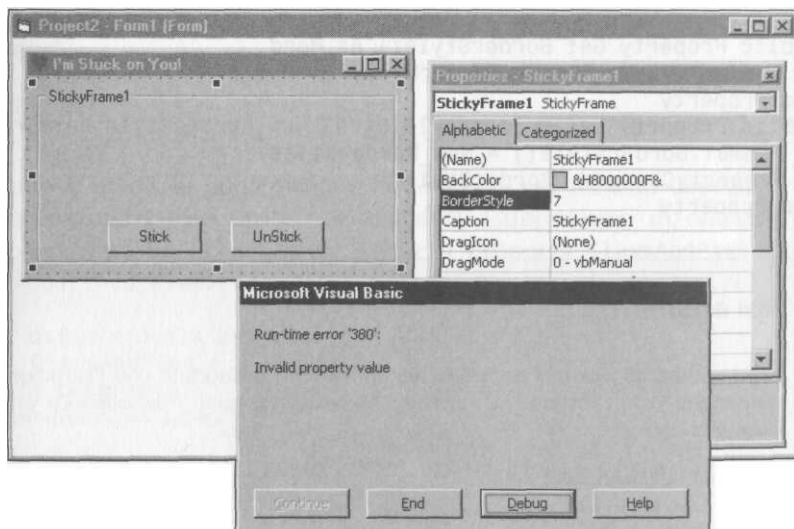
A questo punto, avviando il progetto e facendo clic sul pulsante *cmdStick,* verrà richiamato il metodo Stick del controllo StickyFrame e il cursore resterà intrappolato all'interno del frame fino a quando non farete clic sul pulsante *cmdUnStick* (Figura 26.5). Questa è la dimostrazione di come sia realmente molto semplice creare nuovi controlli, come StickyFrame, in grado di estendere le capacità dei controlli preesistenti.

Figura 26.5

Il cursore rimane intrappolato all'interno di StickyFrame.

Proprietà di tipo enumerato

Sperimentando con l'interfaccia del controllo StickyFrame (ovvero inserendo valori nella finestra *Properties* dopo aver selezionato il controllo) vi renderete conto che alcune proprietà non funzionano esattamente come si vorrebbe. Un ottimo esempio è rappresentato dalla proprietà BorderStyle, che serve per specificare se il frame deve avere o meno un bordo. Nel controllo Frame standard, la proprietà BorderStyle ha due possibili valori: 0-None e 1-Fixed Single. In questo modo lo sviluppatore che utilizza il controllo può scegliere uno dei due valori utilizzando una lista contenuta nella finestra *Properties.*
Ciò nonostante, il controllo StickyFrame non presenta la stessa interfaccia nella finestra *Properties;* BorderStyle accetta semplicemente un valore numerico intero che deve essere digitato dall'utente. Se l'utente digita un numero diverso da O oppure 1, si verifica un errore runtime, come mostrato in Figura 26.6.

Figura 26.6

Assegnando a BorderStyle un valore diverso da 0 o 1 si genera un errore.

L'errore visualizzato in Figura 26.6, Error 380, significa che è stato assegnato alla proprietà un valore errato. Ovviamente, sarebbe preferibile che l'utente non fosse in grado di generare un simile errore. Per scongiurare questo rischio, è necessario sapere come presentare una lista a discesa di possibili valori. Di seguito viene presentalo il codice che è stato creato da ActiveX Control Interface Wizard per le procedure delle proprietà di StickyFraine, incluse le routine di delega dal controllo costituente:

```
Public Property Get BorderStyle() As Integer
   BorderStyle   =  Frame1.BorderStyle
End Property
Public Property Let BorderStyle(ByVal New_BorderStyle As Integer)
   Frame1.BorderStyle()  =   New_BorderStyle
   PropertyChanged "BorderStyle"
End Property
```

Il problema, in questo caso, è rappresentato dalla dichiarazione di tipo As Integer
Ciò che servirebbe è, invece, un tipo enumerato.
Un tipo enumerato, contenente i due possibili valori per la proprietà BorderStyle, deve essere dichiarato come Public nel modulo UserControl:

```
Public Enum Bord
   None = 0
   FixedSingle = 1
End Enum
```

Inoltre bisogna modificare il tipo di dato restituito dalle procedure Property Get e Let in modo che restituiscano e impostino il valore delle costanti Bord invece di un numero intero:

```
Public Property Get BorderStyle() As Bord
   BorderStyle = Frame1.BorderStyle
End Property
Public Property Let BorderStyle(ByVal New_BorderStyle As Bord)
   Frame1.BorderStyle() = New_BorderStyle
   PropertyChanged "BorderStyle"
End Property
```

Tipi enumerati

I tipi enumerati devono essere dichiarati a livello di modulo con l'istruzione Enum, e consìstono in un insieme di membri, ognuno dei quali è associato a una costante long. Per esempio:

```
Enum WildAnimals
   Lions = 0
   Tigers = 1
   Bears = 2
End Enum
```

Salvate ora il modulo UserControl modificato, inserite un'istanza del controllo all'interno di una form e osservate la proprietà BorderStyle nella finestra *Properties.* Come mostrato in Figura 26.7, ora la proprietà BorderStyle può essere impostata utilizzando la casella di riepilogo a discesa definita dal tipo enumerato Bord.

Figura 26.7

Lista a discesa generata dal tipo enumerato.

Proprietà enumerate personalizzate

È molto semplice creare proprietà personalizzate che vengano visualizzate sotto forma di caselle di riepilogo a discesa nella finestra *Properties* del controllo. Supponiamo, per esempio, di voler creare una proprietà ClawsAndPaws che possa assumere un valore scelto dall'utente da una casella di riepilogo a discesa. Prima di tutto si rende necessario definire una enumerazione per la casella a discesa :

```
Public Enum WildAnimals
   Lions = 0
   Tigers = 1
   Bears = 2
End Enum
```

A questo punto è possibile implementare la proprietà nel modulo UserControl come di consueto, attraverso procedure Property (supponiamo di omettere la parte di implementazione della persistenza della proprietà ClawsAndPaws negli eventi InitProperties, ReadProperties e WriteProperties di UserControl) :

```
 'Valori di default della proprietà:
Const m_def_ClawsAndPaws = 2 'Bears
 'Variabili di proprietà:
Dim m_ciawsAndPaws As Long

Public Property Get ClawsAndPaws() As WildAnimals
   ClawsAndPaws = m_ClawsAndPaws
EndProperty

Public Property Let ClawsAndPaws(ByVal _
   New_dawsAndPaws As WildAnimals)
   m_ClawsAndPaws = New ClawsAndPaws
```

```
    PropertyChanged "ClawsAndPaws"
End Property
```

Così facendo, la proprietà ClawsAndPaws viene visualizzata sotto forma di una li a discesa nella finestra *Properties,* come mostrato nella Figura 26.8.

Figura 26.8 *Le proprietà personalizzate possono essere selezionate con caselle di riepilogo a discesa.*

Impostazione di una proprietà predefinita

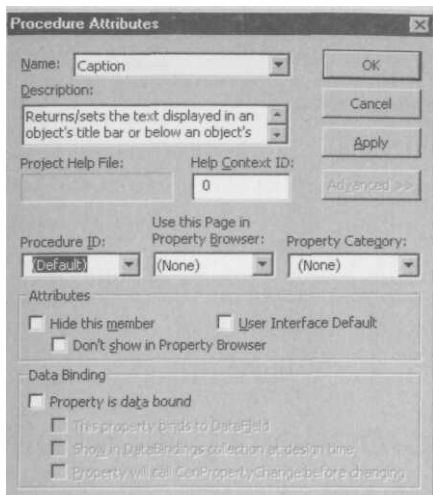
La proprietà predefinita di un controllo è quella utilizzata implicitamente quando non viene specificata in maniera esplicita. Per esempio, la proprietà Text è la proprietà predefinita del controllo casella di testo. Quindi il comando:

```
Text1 = "The more it SNOWS-tiddely-pom"
```

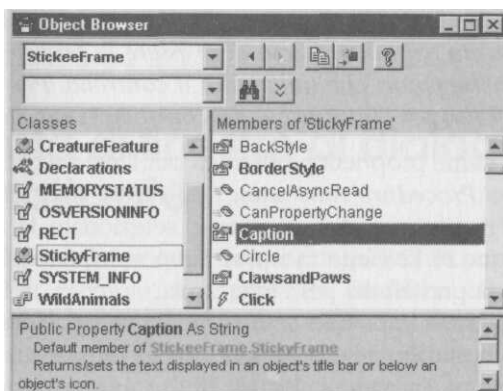
è identico, nell'effetto, al comando

```
Text1.Text = "The more it SNOWS-tiddely-pom"
```

Infatti entrambi i comandi assegnano la stringa alla proprietà Textl .Text. La proprietà predefinita deve essere quella che, con buona probabilità, verrà utilizzata più di frequente, nel codice, da parte degli utenti del controllo. Per impostare la proprietà (o il metodo) predefiniti per un controllo è sufficiente utilizzare la finestra di dialogo *Procedure Attributes,* alla quale è possibile accedere dal menu *Tools* di VB quando è aperta la finestra *Code Editar.* Nella Figura 26.9 viene mostrata l'impostazione come proprietà predefinita della proprietà Caption del controllo Sticky-Frame, attraverso la casella di riepilogo *Procedure ID.*

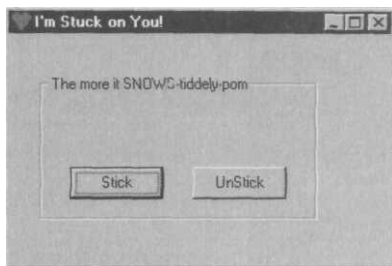
Figura 26.9
Impostazione della proprietà predefinita del controllo.

Se esiste già un membro predefinito, è necessario impostarne a (None) *il valore corrispondente nella lista* Procedure ID *prima di poter impostare a* (Default) *il* Procedure ID *di un altro membro.*

La finestra *Obj'ect Browser,* mostrata nella Figura 26.10, evidenzia il membro predefinito di una classe. Nella figura è possibile intuire che Caption è il membro predefinito per la classe StickyFrame grazie alla piccola pallina presente sopra la relativa icona (sulla sinistra). Il fatto che il membro sia quello predefinito è evidenziato anche nella descrizione del membro nel riquadro presente nella parte bassa della finestra della finestra *Object Browser.*

Figura 26.10
Individuazione del membro predefinito di una classe.

Una volta che il membro è stato impostato come predefinito, è possibile ometterne tutti i riferimenti espliciti nel codice. Per esempio:

```
Private Sub Form_Load()
   StickyFramel = "The more it SNOWS-tiddely-pom"
End Sub
```

imposta la proprietà Caption del controllo StickyFrame, producendo il risultat mostrato nella Figura 26.11.

Figura 26.11 *Omissione di riferimenti espliciti a un membro predefinito.*

Creazione di una proprietà predefinita per l'interfaccia utente

Quando una nuova istanza di un controllo viene posizionata in un contenitore, Visual Basic sceglie una proprietà da evidenziare nella finestra *Properties.* Questa sarà l'ultima proprietà che è stata evidenziata nella finestra *Properties,* a patto che il nuovo controllo selezionato abbia tale proprietà. In caso contrario, Visual Basic utilizza la proprietà che lo sviluppatore ha impostato come proprietà *predefinita per l'interfaccia.* Se lo sviluppatore non ha operato una scelta esplicita in questo senso, Visual Basic ne sceglie semplicemente una in base all'ordine in cui i membri sono stati aggiunti al controllo.

La proprietà predefinita per l'interfaccia (che, ovviamente, è una cosa differente dalla proprietà predefinita del controllo) definisce anche la procedura visualizzata nella finestra Code Editor *quando questa viene apena dopo aver selezionato* UserControl. *La proprietà predefinita per l'interfaccia deve essere la proprietà che viene più spesso impostata dagli sviluppatori che utilizzano il controllo. Per esempio, ha senso che la proprietà predefinita per l'interfaccia del controllo* Timer *sia* Interval.

Per impostare una proprietà come proprietà predefinita per l'interfaccia bisogna utilizzare la finestra di dialogo *Procedure Attributes;* dopo aver selezionato la proprietà tramite la casella di riepilogo a discesa *Name,* selezionate la casella *User Interface Default.* Nella Figura 26.12 viene mostrata l'impostazione della proprietà ClawsAndPaws come proprietà predefinita per l'interfaccia.
Una volta che una proprietà è stata impostata come predefinita per l'interfaccia, creando una nuova istanza del controllo, tale proprietà verrà evidenziata, per default, nella finestra *Properties,* come mostrato nella Figura 26.13.

Figura 26.12 *Impostazione di una proprietà come predefinita per l'interfaccia.*

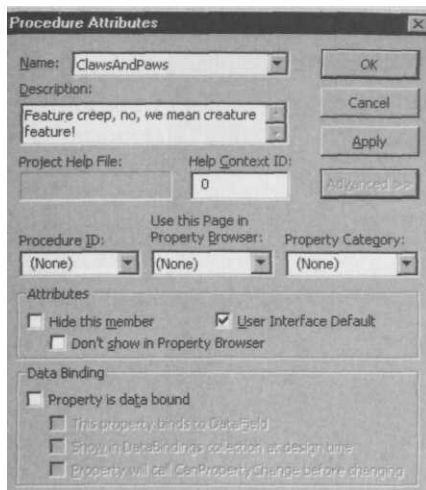
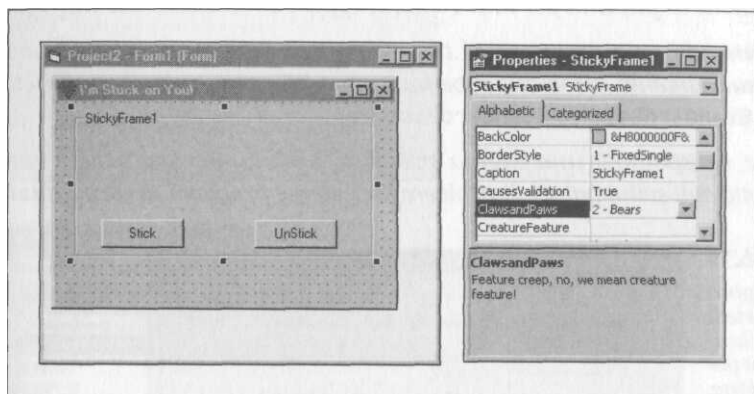
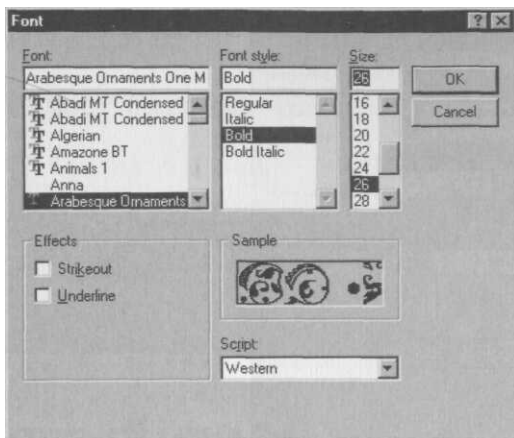



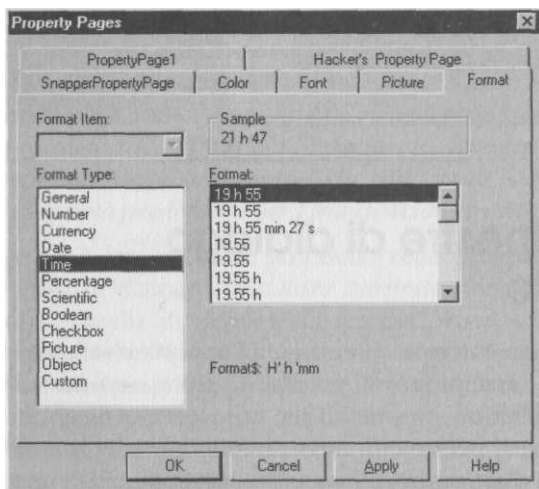
Figura 26.13 *Evidenziazione della proprietà predefinita per l'interfaccia della nuova istanza.*

Aggiunta di finestre di dialogo personalizzate

In alcuni casi, una proprietà è troppo complessa da impostare utilizzando la finestra *Properties*. Per esempio, una proprietà potrebbe essere essa stessa un oggetto, dotato, a sua volta, di ulteriori proprietà. Per un esempio di questa situazione, potete provare a fare clic sul pulsante di espansione presente a lato della proprietà Font per aprire la pagina delle proprietà associata, mostrata nella Figura 26.14.

Figura 26.14
La proprietà Font è un oggetto dotato, a sua volta, di ulteriori proprietà.

Una proprietà può addirittura consistere di una collezione di oggetti (per esempio un insieme di pulsanti per una Toolbar, oppure i nodi di un struttura ad albero).

Dichiarando una proprietà di tipo Font, OLE_COLOR *oppure* Picture, *essa verrà automaticamente associata rispettivamente alla pagina delle proprietà* StandardFont, StandardColor *o* StandardPicture.

Inoltre, è disponibile una pagina StandardFormat, *mostrata nella Figura 26.15, che permette agli sviluppatori di risolvere facilmente problemi di formattazione.*

Figura 26.15
La pagina delle proprietà Standard Format per la formattazione.

Per associare una pagina delle proprietà a una proprietà personalizzata, è necessario innanzitutto creare il corrispondente file .Pag. Per fare un esempio, aggiungete una pagina personalizzata chiamata CreatureFeature, salvata come Creature.Pag e associata al progetto di controllo StickyFrame. Poi aprite la finestra di dialogo *Procedure Attributes* e utilizzate la casella di riepilogo a discesa *Use this Page* in *Pro-*

perty Browser per collegare una proprietà personalizzata alla pagina delle proprietà appena creata. (Nella Figura 26.16 viene mostrato come associare la proprietà CreatureFeature alla pagina personalizzata CreatureFeature.)

Ora, aprendo la finestra *Properties* dopo aver selezionato un'istanza del controllo StickyFrame, la proprietà CreatureFeature sarà rappresentata con un tasto "..." accanto alla relativa colonna Value, come mostrato nella Figura 26.17.

Figura 26.16 *Connessione di una proprietà a una pagina delle proprietà.*

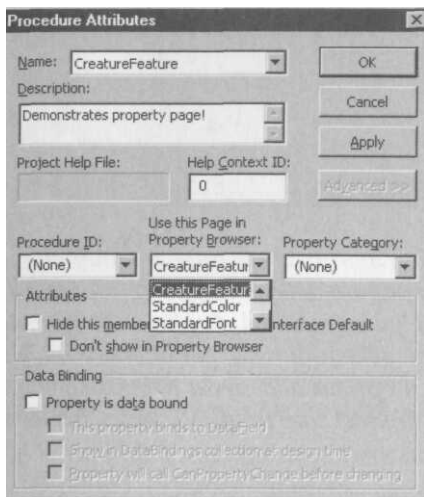



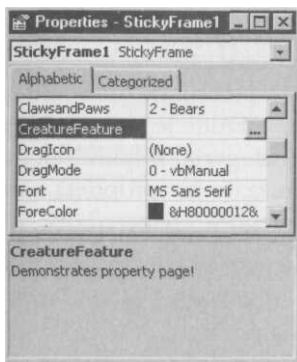
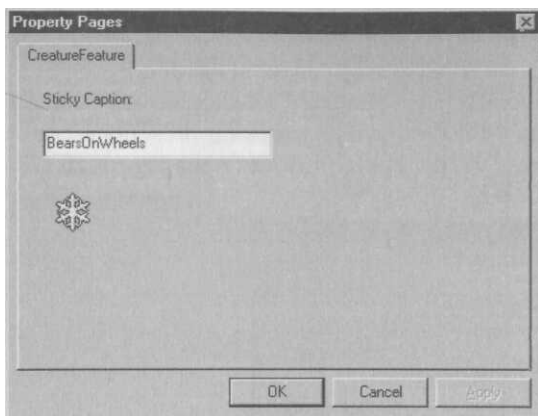
Figura 26.17 *Il pulsante "..." indica che la proprietà è legata a una pagina delle proprietà.*

Facendo clic sul pulsante "…", verrà aperta la pagina delle proprietà relativa, come mostrato nella Figura 27.18.

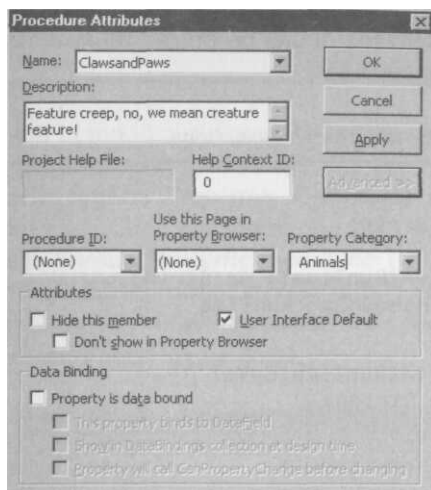
Figura 26.18
La Property Page può essere disegnata liberamente.

Naturalmente, dovrete occuparvi di sincronizzare tutte le modifiche apportate alle proprietà mostrate all'interno detta pagina dette proprietà con le corrispondenti proprietà del controllo (consultate il paragrafo del Capitolo 25 che descrive come aggiungere manualmente le pagine delle proprietà, per ottenere maggiori informazioni). È consentito associare più proprietà alla stessa pagina personalizzata e, in effetti, questo è un ottimo modo di organizzare proprietà legate tra loro.

Raggruppamento di proprietà per categoria

Una nuova caratteristica di VB6 consiste nel presentare due differenti schede all'interno della finestra *Properties,* una che mostra tutte le proprietà del controllo e l'altra che presenta le proprietà suddivise per categoria. È quindi possibile, in fase di progettazione di un nuovo controllo, organizzare le proprietà assegnandole a una delle categorie esistenti oppure a nuove categorie create ad hoc.

Organizzare le proprietà in categorie è sicuramente una buona idea, dato che Visual Basic posiziona all'interno di una generica categoria Misc *tutte le proprietà alle quali non è stata assegnata una categorìa specifica.*

Come prevedibile, per associare le categorie alle proprietà si utilizza la finestra di dialogo *Procedure Attrìbutes* e, per essere più precisi, la casella di riepilogo a discesa *Property Category;* per assegnare alla proprietà una categoria già esistente è sufficiente scegliere tra le voci già contenute nella lista, mentre per creare una nuova categoria basta scriverne il nome, sempre nella casella della lista a discesa (come e stato fatto, nell'esempio mostrato nella Figura 26.19, per la categoria *Animals).*

Figura 26.19 *Utilizzo di Property Category per selezionare o creare categorie.*

Per assegnare più proprietà alla stessa categoria personalizzata è però necessario digitare a mano, ogni volta, il nome della categoria (nell'esempio, Animals), *perché le categorie personalizzate non vengono aggiunte alla casella di riepilogo.*

Una volta che le proprietà saranno state associate alle rispettive categorie mediante la casella di riepilogo *Property Category,* appariranno organizzate per categoria all'interno della finestra di dialogo *Properties,* come si vede nella Figura 26.20.

Figura 26.20 *Risultato dell'assegnazione delle proprietà alle categorie.*

Proprietà in fase di progettazione e in fase di esecuzione

In alcuni casi può essere utile creare proprietà che siano di sola lettura in fase di esecuzione e che possano essere impostate solo in fase di progettazione. Per raggiungere Questo obiettivo è sufficiente creare la proprietà personalizzata nel modo consueto e controllare il valore della proprietà `User-Mode` dell'oggetto `Ambient` all'interno delle Procedure `Property Let` o `Property Set`. (Per maggiori informazioni sull'utilizzo

dell'oggetto Ambient consultate il paragrafo relativo del Capitolo 24.) Il Listato 26 mostra l'implementazione in UserControl di una proprietà (ImReadOnlyAtRunTime) che risulta di sola lettura in fase di esecuzione:

Listato 26.1 *Una proprietà in sola lettura in fase di esecuzione.*

```
Dim m_ImReadOnlyAtRunTime As String

Public Property Get ImReadOnlyAtRunTime() As String
   ImReadOnlyAtRunTime = m_ImReadOnlyAtRunTime
End Property

Public Property Let ImReadOnlyAtRunTime(ByVal _
   New_ImReadOnlyAtRunTime As String)
   If Ambient.UserMode Then
      Err.Raise Number:=31013, _
         Description:= _
         "In esecuzione, la proprietà è di sola lettura."
   End If
   m_ImReadOnlyAtRunTime = New_ImReadOnlyAtRunTime
   PropertyChanged "ImReadOnlyAtRunTime"
End Property
```

Se lo preferite, potete chiaramente implementare questa caratteristica evitando che venga generato un errore nel caso in cui il codice tenti di impostare la proprietà.

Creazione di proprietà valide solo in fase di esecuzione

È possibile creare una proprietà che sia utilizzabile solo in fase di esecuzione modificando le procedure di gestione della proprietà in modo che non permettano la scrittura al suo interno se la proprietà UserMode dell'oggetto Ambient ha valore False. Il Listato 26.2 mostra un esempio di implementazione di una proprietà valida solo in fase di esecuzione:

Listato 26.2 *Creazione di una proprietà valida solo in fase di esecuzione.*

```
Public Property Get ImRunTimeOnly() As String
   If Ambient.UserMode Then
      ImRunTimeOnly = m_ImRunTimeOnly
   End If
End Property

Public Property Let ImRunTimeOnly(ByVal _
   New_ImRunTimeOnly As String)
   If Ambient.UserMode Then
      m_ImRunTimeOnly = New_ImRunTimeOnly
```

```
        PropertyChanged "ImRunTimeOnly"
    End If
End Property
```

Se si fosse fatto in modo che, invece di limitarsi a non fare nulla, la proprietà fallisse in fase di progettazione generando un errore, VB non l'avrebbe visualizzata all'interno della finestra *Properties.* Così come è nell'esempio, infatti, la proprietà `ImRunTimeOnly` è visibile nell'interfaccia del controllo, ma i valori che le vengono assegnati nella finestra *Properties* non sono persistenti. In questo caso, per rimuovere la proprietà dall'interfaccia in fase di progettazione è sufficiente selezionare la casella *Don 't Show in Property Browser* della finestra di *dialogo Procedure Attributes,* come mostrato nella Figura 26.21.

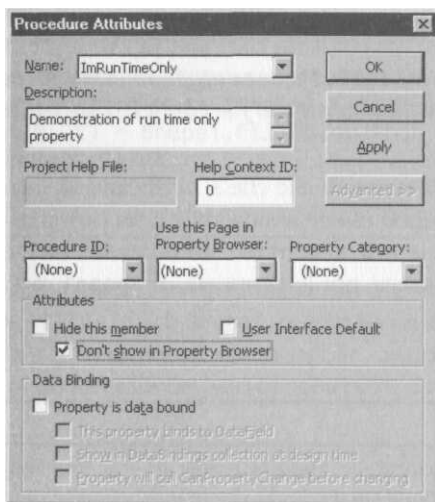
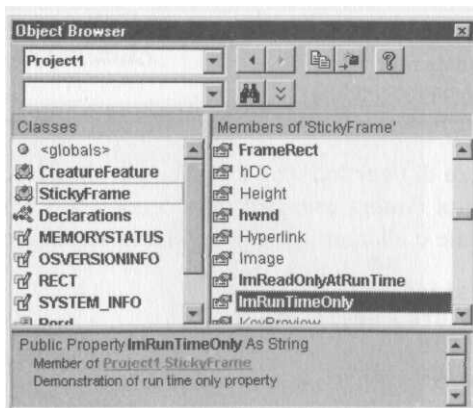
Figura 26.21 *Eliminazione di una proprietà dalla finestra Properties.*

È comunque da notare che, se la casella *Hide this Member* non viene selezionata (vedere Figura 26.21), la proprietà verrà comunque visualizzata nella finestra *Object Browser,* come mostrato in Figura 26.22.

Figura 26.22 *La proprietà appare comunque nell'Object Browser se non si seleziona Hide this Member.*

A questo punto è consigliabile verificare che la proprietà ImRunTimeOnly si comporti correttamente in fase di esecuzione. Per raggiungere lo scopo è sufficiente aggiungere qualche riga di codice nel progetto di test:

```
Debug.Print  StickyFrame1.ImRunTimeOnly
StickyFrame1.ImRunTimeOnly = "Frodo Baggins is a Hobbit"
Debug.Print  StickyFrame1.ImRunTimeOnly
```

Creazione di un controllo basato su più controlli costituenti

Accade molto spesso di creare un nuovo controllo basandosi su più controlli costituenti. È possibile, per esempio, combinare un controllo Label *e un controllo* Shape *per creare un controllo che presenti un'etichetta rotonda. (Il controllo, il progetto di test e il file sorgente del controllo* circButton *relativi all'esempio presentato di seguito si trovano sul CD-ROM rispettivamente nei file CircleB.Vbp, tstCirc. Vbp e CircleB.Ctl.)*

Se lo UserControl rispondesse agli eventi di clic che si verificano all'interno dell'area del cerchio, il nuovo controllo potrebbe essere assimilabile a un pulsante rotondo. Per creare il nuovo controllo, aggiungete a UserControl un controllo Shape (Shape1) e un controllo Label (Label1). Impostate poi le proprietà di Shapel e Label"! come indicato nelle Tabelle 26.1 e 26.2.

Tabella 26.1 *Proprietà di Shapel:.*

Proprietà	Valore
BorderStyle	0-Transparent
FillColor	&HOOOOOOFF (Red)
FillStyle	0-Solid

Tabella 26.2 *Proprietà di Lateli.*

Proprietà	**Valore**
Alignment	2-Center
BackStyle	0-Transparent
ForeColor	&HOOFFFFFF (White)

Aggiungete poi all'evento Resize di UserControl il codice per dimensionare il controllo Shape in modo che occupi l'intera area del nuovo controllo circButton, e per centrare l'etichetta in verticale e allargarla in base alla larghezza di circButton:

```
Private Sub UserControl_Resize()
   Shape1.Move 0,  0,  ScaleWidth,  ScaleHeight
   Label1.Move 0, (ScaleHeight _
       -  Label1.Height)  /  2,  ScaleWidth
End Sub
```

povrete poi implementare la delega dei membri nel modo consueto. A questo punto, però, sorge un problema con l'evento Click; infatti non è difficile delegare gli eventi Click innescati dal controllo Label in modo che vengano elaborati da circButton:

```
Private Sub Label1_Click()
   RaiseEventClick
End Sub
```

In questo modo l'evento Click di circButton viene innescato ogni volta che l'utente fa clic sulla Label; il problema è che, idealmente, l'evento clic dovrebbe essere innescato anche quando l'utente fa clic sull'oggetto Shape, il quale però non possiede un evento Click. Il codice del Listato 26.3, relativo alla gestione dell'evento MouseUp di UserControl, risolve il problema sfruttando una tecnica chiamata ***bit-testing.***

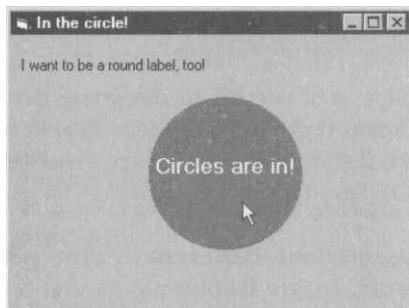
Listato 26.3 *Implementazione dell'evento Click tramite bit-testing.*

```
Private Sub UserControl_MouseUp(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)
   If Point(X, Y) = Shape1.FillColor Then
      RaiseEvent Click
   End If
End Sub
```

A patto che non vengano inseriti, all'interno di UserControl, altri controlli dotati dello stesso colore FillColor di Shapel, questa tecnica funzionerà correttamente. Per mettere alla prova il controllo è ora possibile compilarlo, aprire un nuovo progetto e aggiungere in un form il controllo circButton, assieme al codice appropriato:

```
Private Sub circButton1_Click()
   Form1.Caption = "In the circle!"
End Sub

Private Sub Form_Click()
   Form1.Caption = "Out of the circle!"
End Sub
```

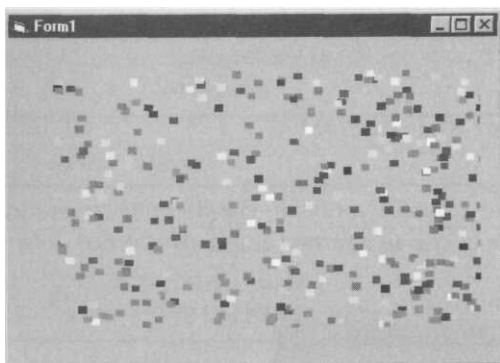
Avviando il programma potrete quindi verificare che il controllo circButton innesca l'evento solo quando l'utente fa clic all'interno (e non all'esterno) del cerchio, come mostrato in Figura 26.23.

Figura 26.23 *Esempio di controllo creato a partire da più controlli costituenti.*

Controlli user-drawn

Un controllo *user-drawn è* un controllo che "disegna" autonomamente il proprio aspetto (e, di conseguenza, il modo in cui si presenta non dipende dall'aspetto di alcun controllo costituente). Generalmente l'unico punto in cui si inserisce codice per disegnare l'aspetto del controllo è la procedura di gestione dell'evento Paint di UserControl; tale codice può richiamare metodi grafici dell'oggetto UserControl oppure funzioni delle API di Windows, se necessario.
È importante determinare quando il controllo deve essere disegnato, lo stato in cui si trova (per esempio, cliccato oppure non cliccato) e se si rende necessario disegnare un rettangolo di focus oppure no. (Per i controlli creati a partire da controlli costituenti, la maggior parte di questi dettagli vengono gestiti automaticamente.)
Quando il contenitore ridisegna l'area nella quale si trova il controllo, viene automaticamente generato l'evento Paint dell'oggetto UserControl. Se l'aspetto del controllo deve cambiare in base alle azioni dell'utente (per esempio, se l'utente fa clic sul controllo), è possibile generare l'evento Paint richiamando il metodo Refresh dell'oggetto UserControl.

Creazione di un controllo "Coriandoli"

Per sperimentare direttamente la creazione di un controllo user-drawn, creeremo ora un controllo Confetti, *che si presenterà, in fase di esecuzione, come mostrato in Figura 26.24. Il modulo del controllo* Confetti *si trova nel file Confetti.Ctl, mentre il progetto ActiveX Control che include il controllo fa pane del progetto Confetti. Vbp. È inoltre disponibile un progetto di test chiamato tstConf. Vbp.*

Figura 26.24
Esempio di controllo user-drawn.

Il controllo Confetti è molto semplice, e si occupa di disegnare casualmente un numero di coriandoli colorati specificato dalla sua proprietà Iterations. Questo accade ogni volta che viene innescato il suo evento Paint (per esempio quando il controllo viene ridimensionato o inizializzato) oppure quando viene richiamato il suo metodo Refresh.
Uno sviluppatore può forzare la generazione dell'evento Paint richiamando il metodo Refresh del controllo Confetti. Inoltre il controllo ha una proprietà Ena-

bled che permette allo sviluppatore di disattivare il comportamento potenzialmente noioso del controllo in casi particolari. È stato inoltre aggiunto del codice per fare in modo che il controllo svolga il proprio compito solamente in fase di esecuzione. Il Listato 26.4 mostra tutto il codice del modulo `UserControl` necessario per creare e far funzionare il controllo:

Listato 26.4 *Il controllo "Confetti".*

```
Option Explicit
 'Default Property Values:
Const m_def_Iterations = 5000
Const m_def_Enabled = True

 'Property Variables:
Dim m_Iterations As Long
Dim m_Enabled As Boolean

Public Property Get Iterations() As Long
   Iterations = m_Iterations
End Property

Public Property Let Iterations(ByVal New_Iterations As Long)
   m_Iterations = New_Iterations
   PropertyChanged "Iterations"
End Property

Public Property Get Enabled() As Boolean
   Enabled = m_Enabled
End Property

Public Property Let Enabled(ByVal New_Enabled As Boolean)
   m_Enabled = New_Enabled
   PropertyChanged "Enabled"
End Property

Function Refresh()
   UserControl_Paint
End Function

Private Sub UserControl_InitProperties()
   m_Iterations = m_def_Iterations
   m_Enabled = m_def_Enabled
End Sub

Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
   m_Iterations=PropBag.ReadProperty("Iterations",_
      m_def_Iterations)
   m_Enabled = PropBag.ReadProperty("Enabled", m_def_Enabled)
End Sub

Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
   Call PropBag.WriteProperty("Iterations", m_Iterations, _
      m_def_Iterations)
   CallPropBag.WriteProperty("Enabled",m_Enabled,_
      m_def_Enabled)
End Sub
```

```
Private Sub UserControl_Paint()
    Dim I As Integer, X1 As Integer, Y1 As Integer, Color As Long
    If Enabled Then
        If. Ambient.UserMode Then 'runtime only!
            Randomize
            For I = 1 To Iterations
                X1 = Rnd * ScaleWidth
                Y1 = Rnd * ScaleHeight
                Color = QBColor(Rnd * 15)
                Line (X1, Y1)-(X1 + 85, Y1 + 65), Color, BF
            Next I
        End If
    End If
End Sub
```

A questo punto è possibile inserire il controllo in un contenitore ed eseguire il progetto risultante: una serie di coriandoli colorati riempiranno l'area del controllo (a patto che la sua proprietà Enabled sia impostata a True).

I metodi grafici utilizzati nell'evento Paint *del controllo* Confetti *sono stati spiegati nel paragrafo del Capitolo 16 che descrive gli effetti speciali.*

Che cosa sono gli oggetti UserControl

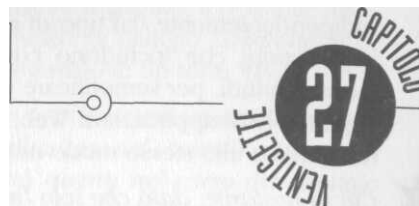
È importante tenere a mente che gli oggetti UserControl non sono form Visual Basic. Infatti, alcuni degli eventi che possono essere utilizzati nei form non sono applicabili a un oggetto UserControl. Per esempio l'oggetto UserControl non ha eventi Activate e Deactivate, dato che i controlli non possono essere attivati o disattivati (mentre ciò è possibile con i form). Inoltre gli eventi Load, Unload e QueryUnload, familiari a chi utilizza i form, mancano nel ciclo di vita di un controllo.
Gli eventi Initialize e ReadProperties di UserControl forniscono la funzionalità dell'evento Load di un form, ma la differenza sostanziale consiste nel fatto che, quando si verifica l'evento Initialize, il controllo non è ancora stato inserito nel contenitore, perciò gli oggetti Extender e Ambient del contenitore non sono ancora disponibili; invece, quando vengono generati gli eventi InitProperties e ReadProperties, il controllo si trova già all'interno del contenitore.
L'evento di UserControl che più si avvicina all'evento Unload del form è l'evento Terminate; durante la sua elaborazione, i controlli costituenti esistono ancora ma non è più possibile accedere al contenitore, perché il controllo non si trova più al suo interno. Inoltre, l'evento WriteProperties non può essere utilizzato al posto di Unload perché viene generato solamente in fase di progettazione.
Gli oggetti UserControl non hanno l'evento QueryUnload, perché i controlli sono solo una parte del form e non è nelle loro possibilità decidere se il form che li contiene deve essere chiuso oppure se il processo di chiusura del form debba essere arrestato, una volta che il form lo ha avviato. Infatti, proprio come avviene per un guerriero Borg, il compito di uno UserControl è di autodistruggersi quando ne riceve l'ordine, senza fare domande.

Gli UserControl *non devono essere necessariamente visibili in fase di esecuzione. (Considerate, per esempio, il controllo* TimerJ *Per ottenere lo scopo è sufficiente impostare la proprietà* Visible *di* UserControl *a* False. *È inoltre possibile utilizzare i metodi di* UserControl *per manipolare l'aspetto di altri oggetti (per esempio, l'aspetto del contenitore all'interno del quale si trova il controllo).*

Riepilogo

Creare controlli ActiveX personalizzati è un'esperienza tremendamente eccitante e gratificante. Questo capitolo ha discusso riguardo molti argomenti utili per iniziare a sviluppare e implementare controlli.

- Avete imparato a creare un controllo StickyFrame.
- Avete scoperto le proprietà enumerate.
- È stato spiegato come creare proprietà enumerate personalizzate.
- È stata descritta la finestra di dialogo *Procedure Attributes.*
- Avete imparato a impostare la proprietà predefinita per il controllo.
- Avete imparato a impostare la proprietà predefinita per l'interfaccia utente.
- Avete imparato a creare finestre di dialogo personalizzate per le proprietà.
- Avete imparato ad associare delle categorie alle proprietà.
- È stato spiegato come creare proprietà in sola lettura.
- È stato descritto come creare proprietà accessibili solo in fase di esecuzione.
- Avete imparato a creare un controllo a partire da più controlli costituenti.
- Sono stati descritti i controlli *user-drawn.*
- Avete creato un controllo Confetti.
- È stata spiegata la differenza tra UserControl e form di Visual Basic.

CONTROLLI ACTIVEX INSTALLATI VIA WEB

- Installazione di un controllo attraverso il Web
- Verifica del funzionamento di un controllo in Internet Explorer
- Package and Deployment Wizard
- Rendere sicuri i controlli per l'utilizzo con lo scripting
- Utilizzo dei file creati da Package and Deployment Wizard
- Utilizzo di un controllo ActiveX su Web

Una volta utilizzato Visual Basic 6 per creare un controllo ActiveX, questo può essere utilizzato da qualsiasi applicazione che metta a disposizione un contenitore in grado di ospitare controlli ActiveX, e un'applicazione particolarmente eccitante di questo concetto consiste nell'utilizzare controlli ActiveX in applicazioni Web Internet (pubbliche) e intranet (Private).
In ogni caso, non tutti i contenitori di controlli ActiveX sono uguali e, in particolare, una pagina Web si comporta in maniera leggermente diversa da un form Visual Basic. Per poter fare in modo che un'applicazione Web, che include un controllo, funzioni correttamente, il controllo ActiveX deve essere installato sulla macchina dell'utente dell'applicazione. Ciò significa che, se il controllo non è presente sulla macchina, esso deve essere scaricato dal server Web e installato. Questo capitolo spiega come utilizzare i controlli ActiveX, creati con Visual Basic, in ambiente Web.

Installazione di controlli attraverso il Web

Internet è una rete Web pubblica (ovvero è una vasta rete di computer connessi attraverso protocolli TCP/IP e HTTP e accessibile a chiunque sia dotato del software appropriato). Ogni nodo della rete (i nodi vengono anche chiamati server Web) pubblica contenuti utilizzando il linguaggio HTML (e le estensioni di HTML come i controlli ActiveX incorporati). I browser sono applicazioni client che decodificano il codice HTML generato dal server Web e lo presentano sotto forma di una pagina formattata.
Una rete Web intranet è concettualmente la stessa cosa di Internet, con l'unica differenza che l'accesso alla rete è controllato (ossia la rete è privata).

Esiste poi una terza variante, *extranet,* ossia una rete Web privata (una intranet) resa disponibile al pubblico solo in contesti specifici. Per esempio, una società che svolga attività di corriere per merci potrebbe rendere disponibili al pubblico una parte delle proprie interfacce intranet, per permettere ai clienti di tenere sotto controllo le proprie consegne. In questo caso l'azienda ha trasformato la propria intranet in una extranet.

Indipendentemente dal tipo di rete Web utilizzata (extranet, intranet o Internet), le applicazioni che includono controlli ActiveX funzioneranno sempre allo stesso modo. Quindi, per semplificare i concetti descritti in questo capitolo, parleremo più in generale di applicazioni Web, tenendo comunque presente che le stesse tecniche funzionano allo stesso modo nei tre ambienti di rete.

Ciò nonostante, dato che solo Internet Explorer comprende, ed è in grado di utilizzare, i controlli ActiveX (ciò non accade con altri browser concorrenti; per una spiegazione, consultate il riquadro "Browser che 'capiscono' ActiveX" che segue), è sicuramente più pratico utilizzare ActiveX prevalentemente in applicazioni intranet (una rete privata nella quale è possibile specificare il software che deve essere utilizzato per la navigazione).

A questo punto, consideriamo, per esempio, una applicazione che sia residente su un server Web, che includa: codice HTML per definire l'aspetto dell'interfaccia, un controllo ActiveX e comandi VBScript per manipolare i membri esposti del controllo in base all'input dell'utente. Per fare in modo che una simile applicazione funzioni sul browser dell'utente, il controllo ActiveX deve essere installato sul sistema client. A questo punto, quindi, vedremo quali sono i passi necessari per scaricare e installare un controllo attraverso una rete Internet; per raggiungere l'obiettivo, è utile riesaminare quali siano i requisiti per effettuare una normale installazione di un controllo standard.

Browser che "capiscono" ActiveX

È possibile pensare a un'applicazione Web come a un'applicazione client/server nella quale il browser svolge il ruolo del client. I contenuti eseguibili (come, per esempio, i controlli ActiveX) si trovano sul server fino a quando il browser client non vi accede. Però, non tutti i browser "capiscono" lo standard ActiveX, per cui, per fare in modo che un'applicazione che include un controllo ActiveX funzioni correttamente, è necessario utilizzare un browser Internet Explorer 3.0 o superiore. Quindi, creando un'applicazione Web che include controlli ActiveX, è necessario essere coscienti del fatto che gli utenti dovranno installare e usare Internet Explorer per poterla utilizzare.

Installazione normale

Per poter installare un controllo su un sistema, il controllo e tutti i file di supporto (come, per esempio, la libreria runtime di Visual Basic) devono essere già presenti sul sistema destinatario e i file di supporto devono trovarsi dove il sistema è in grado di rintracciarli (ossia, solitamente, nella directory Windows/System oppure

nella directory nella quale si trova il controllo). Inoltre, il controllo deve essere registrato nel sistema (operazione effettuabile avviando Regocs32 o Regsvr32.exe). Per esempio:

```
Regsvr32 C:\Windows\Occache\Confetti.Ocx
```

Una volta che il controllo è stato correttamente installato e registrato, è possibile inserirlo in un contenitore (per esempio utilizzando al finestra di dialogo *Components* di VB per aggiungerlo alla *Toolbox,* e successivamente inserirlo in un form VB).

Visual Basic Package and Deployment Wizard è un'applicazione separata che permette di creare un programma stand-alone di installazione del controllo che svolga automaticamente tutte le operazioni necessarie. Più avanti nel corso del capitolo vedremo come sia possibile utilizzare questo Wizard per automatizzare lo scaricamento via Internet; per informazioni più generiche sull'utilizzo di Package and Deployment Wizard consultate invece il Capitolo 35.

Se si desidera rimuovere manualmente un controllo dal sistema, non è sufficiente cancellare i riferimenti al controllo dal Registro di configurazione. Bisogna infatti avviare anche l'utility Regsvr32 specificando il /lag /u *("u" è l'abbreviazione di "unregister"), Per esempio:*

```
Regsvr32 /u C:\Windows\Occache\Confetti.Ocx
```

Le utility di registrazione sono descritte in dettaglio nel Capitolo 9.

Installazione di un controllo da Web

Concettualmente, tutti i programmi di setup Web funzionano allo stesso modo: un controllo ActiveX viene identificato, all'interno del codice HTML che definisce la pagina, attraverso tag <OBJECT> e </OBJECT> e attraverso il CSLID del controllo (un identificatore unico per l'oggetto). Se il controllo è già installato sul sistema destinatario della pagina, una sua istanza viene automaticamente creata quando il browser elabora i corrispondenti tag <OBJECT> .

Se il controllo ActiveX non è stato ancora installato nel sistema destinatario, si rende però necessario scaricarlo e avviare un programma che si occupi di installarlo. Il parametro Codebase del tag <OBJECT> serve per specificare la locazione del programma di installazione compresso sul server (salvato in formato .Cab, ovvero in un file *cabinet).* Allo stesso modo è possibile scaricare tutti i file di supporto per il controllo ActiveX (come il modulo runtime VB oppure la corrispondente Virtual Machine, Msvbvm60.Dll).

Se non si specifica la locazione dei file sul server, essi vengono scaricati dai siti Web di Microsoft.

Quando un'applicazione crea un'istanza di un controllo ActiveX, gli attributi <PARAM NAME> (ovvero delle coppie di nomi di proprietà e dei relativi valori) vengono passate all'evento InitProperties del controllo utilizzando l'oggetto standard ProPertyBag. Per esempio:

```
<PARAM NAME="Enabled" VALUE="0">
```

A differenza di quanto accade quando si inserisce un controllo ActiveX all'interno di un form Visual Basic e poi si chiude il form, le pagine HTML non salvano le informazioni specificate in fase di progettazione. Quindi un controllo posizionato all'interno di una pagina HTML si comporta come se ogni volta venisse creato ex novo. Ciò comporta che, quando il codice HTML viene elaborato dal browser il controllo presente all'interno della pagina riceve gli eventi Initialize, InitProperties e Resize ma non l'evento ReadProperties.
Per aprire un controllo assegnandogli un valore di proprietà personalizzata persistente che non sia quello predefinito, è quindi necessario aggiungere una coppia <PARAM NAME=. . . VALUE=> al tag relativo all'oggetto controllo. Per esempio, il valore predefinito della proprietà Enabled del controllo Confetti è True. Se si desidera creare un'istanza del controllo con la proprietà Enabled impostata a False, è necessario aggiungere la seguente coppia di valori al tag <OBJECT>:

```
<OBJECT . . .>
   <PARAM  NAME="Enabled"  VALUE="0">
</OBJECT>
```

Verifica del funzionamento di un controllo in Internet Explorer

Facendo girare un controllo in ambiente di progettazione di VB6, Visual Basic genera automaticamente il codice HTML necessario a far funzionare il controllo in Internet Explorer, come mostrato nella Figura 27.1.

Questo rende particolarmente semplice verificare il comportamento del controllo all'interno del browser Internet Explorer, dato che non è necessario lasciare l'ambiente di sviluppo.

Esecuzione di Package and Deployment Wizard

Il modo più semplice per creare un programma di setup Web per i propri controlli ActiveX è quello di utilizzare *Package and Deployment Wizard,* un programma che appartiene al gruppo Microsoft Visual Studio 6.0 Tools, e che viene descritto in dettaglio nel Capitolo 35.
Package and Deployment Wizard genera un pacchetto di installazione (ovvero un file .Cab) e del codice HTML di esempio, adattabile per le esigenze degli utenti. Per creare un setup Web per un controllo ActiveX è necessario avviare Package and Deployment Wizard, selezionare il progetto relativo al controllo e fare clic sul pulsante *Package,* come mostrato nella Figura 27.2.

Figura 27.1
I progetti ActiveX Control vengono eseguiti per default in Internet Explorer.

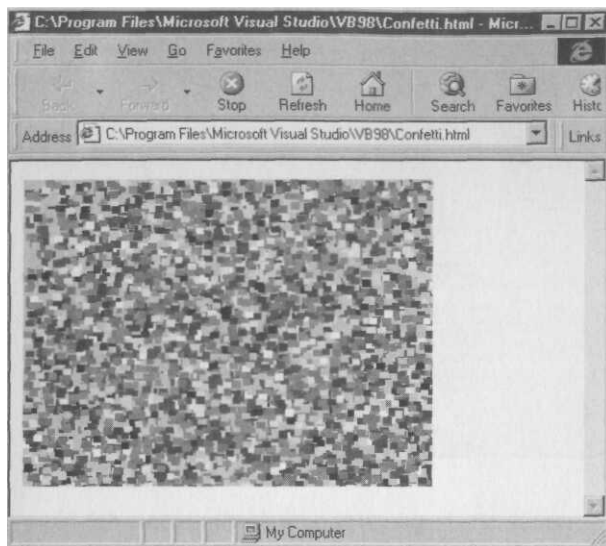
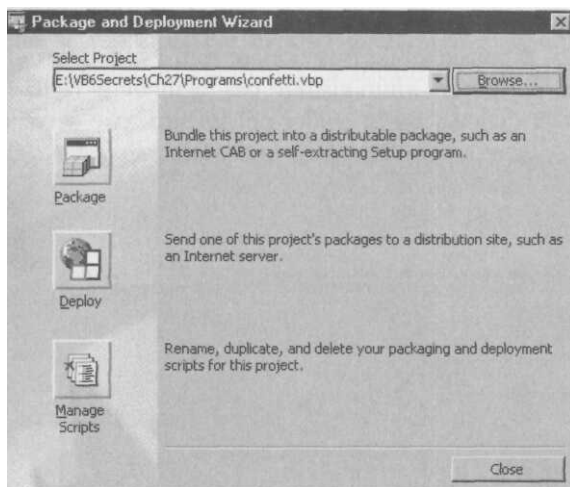



Figura 27.2
Utilizzo del pulsante Package per creare un setup per il controllo.

La routine di setup creata per l'utilizzo via Internet non potrà essere comunque utilizzata per effettuare una normale installazione in locale. Infatti la componente di scaricamento e installazione via Internet è progettata per un utilizzo runtime, e per questo motivo è radicalmente diversa da quella necessaria per l'utilizzo in fase di progettazione.

A questo punto selezionate il tipo di pacchetto che il Wizard dovrà creare (per l'installazione via Web, la scelta corretta è *Internet Package)* come mostrato nella Figura 27.3.

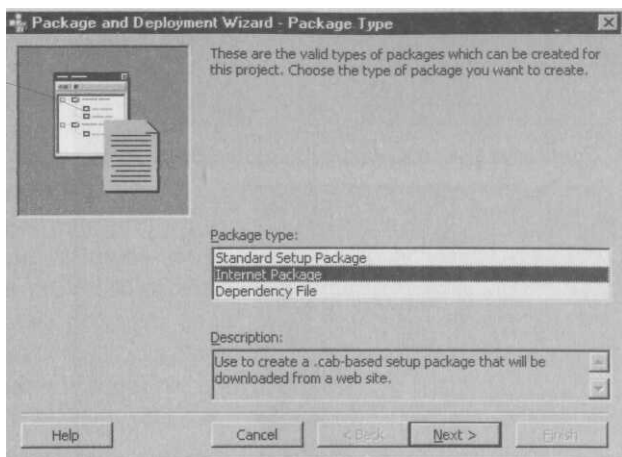
Figura 27.3
Selezione di Internet Package per l'installazione via Web.

È inoltre necessario specificare la posizione di destinazione del file .Cab che verrà creato dal Wizard e dei file di supporto associati. I file .Cab vengono compressi secondo lo standard Microsoft per la consegna. Per esempio, osservando il contenuto del CD-ROM di installazione di Windows, noterete come la maggior parte dei file sia in formato .Cab. È consigliabile posizionare tutti i file .Cab del sito Web in una directory apposita, per poterli amministrare più facilmente. Nella Figura 27.4 è visualizzata la finestra di dialogo *Build Folder* del Wizard, che serve a specificare tale directory, che può essere la directory principale della gerarchia del server Web. In alternativa è possibile copiare in un secondo tempo nella posizione appropriata i file creati dal Wizard.

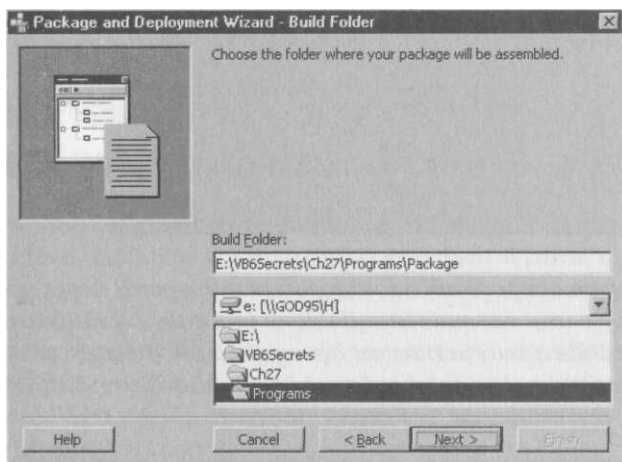
Figura 27.4
Specifica della locazione di destinazione dei file di setup.

La finestra successiva mostra i file che il Wizard ha identificato come necessari per il controllo (vedere Figura 27.5). Tramite la finestra è possibile aggiungere o rimuovere file dalla lista utilizzando le caselle di opzione associate.

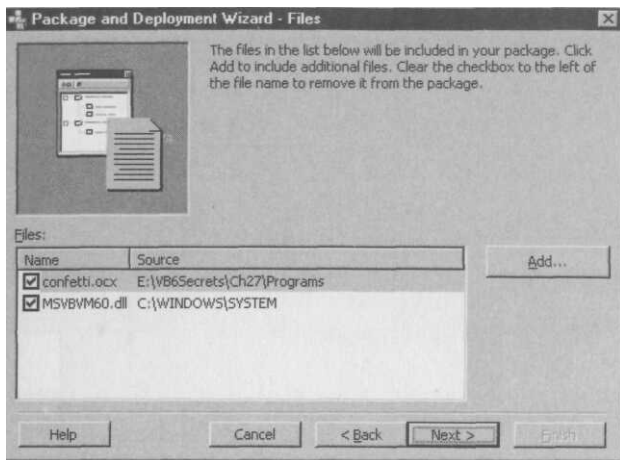
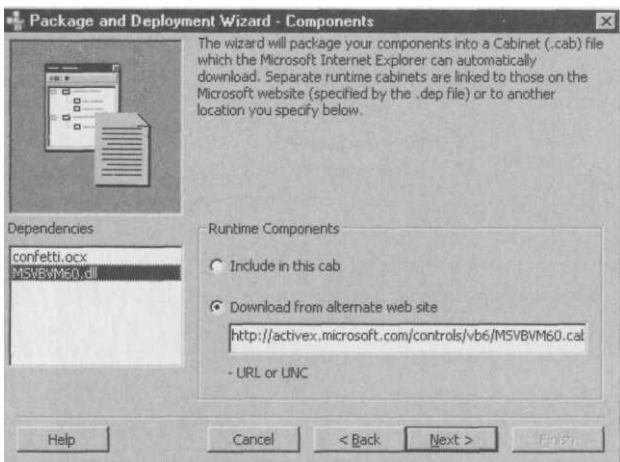
Figura 27.5 *Aggiunta o rimozione di file da distribuire.*

Ogni file della lista può essere incluso direttamente all'interno del file .Cab che verrà generato, oppure, in alternativa, scaricato da una locazione differente della rete Web, come mostrato nella Figura 27.6.

Figura 27.6 *Selezione dei file da includere nel file .Cab.*

Così è possibile fare in modo che alcuni file vengano scaricati automaticamente da uno dei siti Microsoft e assicurarsi che gli utenti ne ottengano sempre la versione più aggiornata.

La finestra successiva del Wizard, mostrata nella Figura 27.7, permette di contrassegnare i controlli come *Safe far Scripting* (sicuri per lo scripting) e *Safe for Initialization* (sicuri per l'inizializzazione). Per maggiori informazioni consultate il paragrafo relativo, più avanti in questo capitolo.

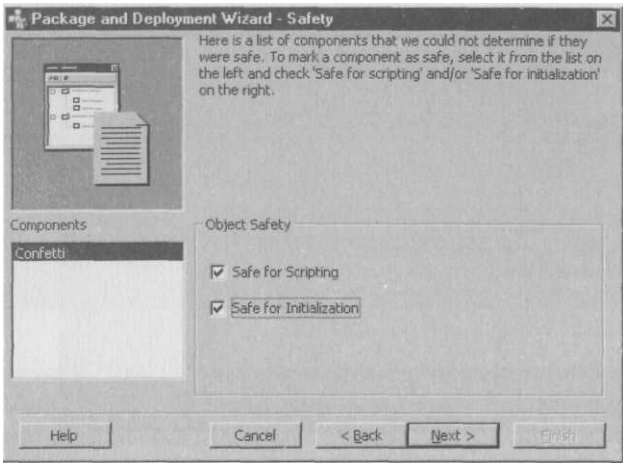
Figura 27.7 *Impostazione di Safe for Scripting e Safe Initialization.*

La locazione di installazione può essere poi personalizzata per ognuno dei file del pacchetto, come mostrato nella Figura 27.8.

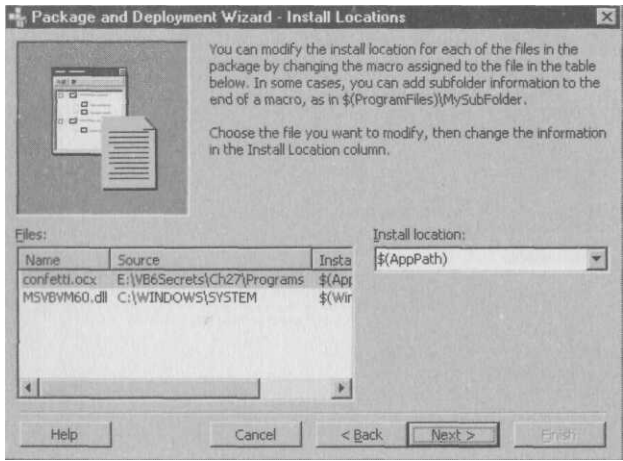
Figura 27.8 *Impostazione della locazione di installazione per ognuno dei file.*

I file che vengono installati come file condivisi (come, per esempio, i file condivisi di libreria o i controlli ActiveX) possono essere rimossi solamente se vengono disinstallati tutti i programmi che li utilizzano. È possibile contrassegnare i file come condivisi nella finestra successiva del Wizard, mostrata nella Figura 27.9.

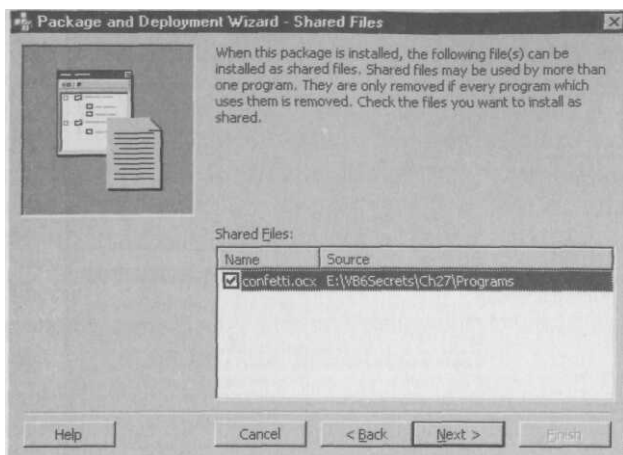
Figura 27.9
Definizione dei file condivisi.

Nell'ultima finestra del Wizard, mostrata nella Figura 27.10, viene infine fornita la possibilità di salvare le opzioni del pacchetto sotto forma di script in modo da poterle riutilizzare in futuro.

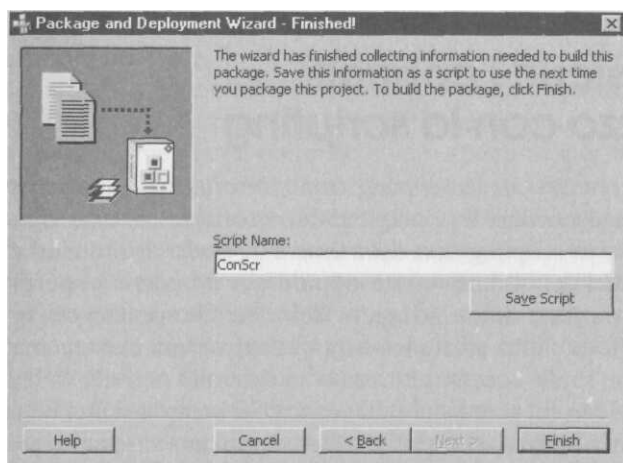
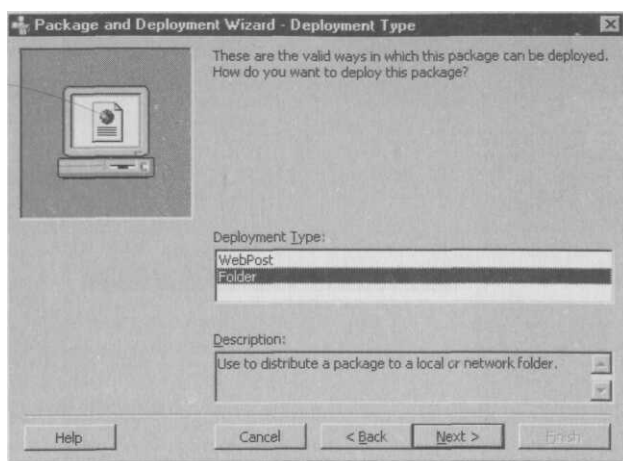
Figura 27.10
Salvataggio delle opzioni sotto forma di script.

La porzione di "Deployment" di Package and Deployment Wizard, può essere avviata dalla finestra principale del Wizard, già mostrata in Figura 27.2, facendo clic sul secondo pulsante, *Deploy,* invece che sul primo, *Package.* Il Deployment consiste nella distribuzione del contenuto di un pacchetto in locale o su uno sito Web. Con la finestra *Deployment Type* del Wizard, mostrata in Figura 27.11, è possibile posizionare i file in una cartella (cioè sul sistema locale o in rete) oppure effettuarne l'invio *(posting)* sul Web.

Figura 27.11 *Distribuzione del pacchetto in locale o su Web.*

Se si indica al Wizard di distribuire il pacchetto su Web, è necessario utilizzare un *provider WebPost* (ovvero un servizio) registrato nel sistema. Esempi di provider WebPost sono FTP, Microsoft Content Replication System, HTTP Post e FrontPage Extended Web.

Rendere sicuri i controlli per l'utilizzo con lo scripting

Rendere sicuri i controlli per lo scripting *(safe far scripting)* è un aspetto da tenere seriamente in considerazione. Il principio di base consiste nel fatto che chi utilizza il controllo attraverso lo scripting non deve essere in grado di sfruttarlo per leggere o scrivere file specifici del sistema ospite oppure per accedere a specifiche voci del Registro di configurazione attraverso la rete Web. (Per file specifici o specifiche voci si intende quelli scelti da chi ha creato lo script.) Infatti, se una persona malintenzionata potesse ottenere un simile accesso attraverso un controllo ActiveX Web, teoricamente nulla di ciò che risiede sul sistema che ha scaricato il controllo sarebbe più al sicuro.
Dato che i controlli ActiveX scaricati da Web non sono altrettanto sicuri quanto i pacchetti in vendita nei negozi, sono stati individuati e implementati numerosi meccanismi per rassicurare gli utenti, come, per esempio, le firme digitali (chiamate anche *certificati di autenticità digitali)* che servono a fornire un percorso utile per rintracciare chi ha creato il controllo attraverso l'azienda che ha fornito il certificato. VeriSign Commerciai Software Publishers è una delle aziende più famose in grado di fornire questo servizio (ed è quella utilizzata da Microsoft per i propri certificati). Utilizzando Package and Deployment Wizard è possibile, durante la creazione del setup Internet per il controllo, includere nel pacchetto un certificato di autenticità.
È ovvio che, per la distribuzione commerciale di controlli ActiveX su Web, è indispensabile ottenere certificati digitali (se non altro per rassicurare i potenziali clienti). ActiveX SDK, per permettere lo sviluppo, mette a disposizione vari certificati digitali di prova, utili sia per effettuare il debugging delle routine di scarica-

mento sia *per* ottenere certificati personalizzati. L'URL per scaricare ActiveX SDK è il seguente : `http://www.microsoft.com/intdev/sdk/sdk.htm`.
È possibile contrassegnare come "sicuro per lo scripting" un controllo, confermando all'utente che non esiste possibilità che uno script presente in una pagina HTML possa causare danni al suo computer, o possa permettere di ottenere informazioni che non sono state fornite volontariamente dall'utente. Un controllo che permette a un programmatore di pagine Web di effettuare una delle seguenti operazioni non può essere definito sicuro per lo scripting:

- Creare un file con un nome specificato all'interno dello script.
- Leggere un file (specificato all'interno dello script) contenuto nell'hard disk dell'utente.
- Inserire informazioni nel Registro di Windows (o in un file .Ini) utilizzando una chiave (o un nome di file) specificato nello script.
- Leggere informazioni dal Registro di Windows (o da un file .Ini) utilizzando una chiave (o un nome di file) specificato nello script.
- Eseguire una funzione di un'API di Windows utilizzando informazioni fornite dallo script.
- Creare o manipolare oggetti esterni utilizzando *programmatic* ID (per esempio "Excel.Application") specificati nello script.

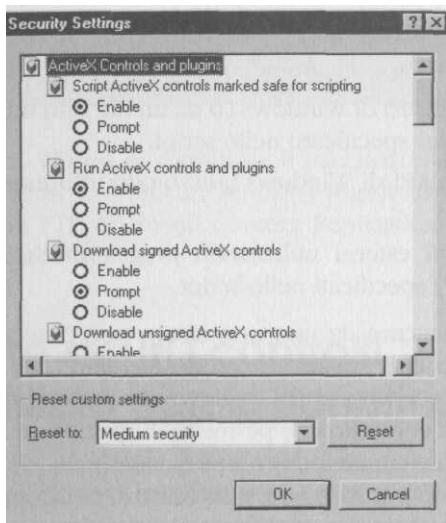
La linea che divide un controllo sicuro da uno non sicuro non è necessariamente ovvia. Per esempio, un controllo che utilizza il metodo `SaveSetting` per scrivere informazioni all'interno della propria chiave del Registro non è da considerarsi non sicuro per lo scripting, mentre un controllo che permette di specificare la chiave del Registro (attraverso una proprietà o un metodo) non è sicuro.
Un controllo che utilizza un file temporaneo può essere sicuro per lo scripting. Però se il nome del file temporaneo può essere specificato dallo script, il controllo non sarà più sicuro, e lo stesso accadrebbe per un controllo in grado di manipolare la quantità di informazioni memorizzabili in un file temporaneo, perché uno script potrebbe continuare a inserire informazioni nel file temporaneo fino a quando l'hard disk dell'utente non fosse completamente pieno.
Come ultimo esempio, un controllo che utilizza chiamate alle API non è necessariamente non sicuro per lo scripting, supponendo però che il controllo permetta allo script di fornire dati alla API e non effettuasse alcun controllo sulle dimensioni di tali dati, un blocco di informazioni troppo grande potrebbe sovrascrivere porzioni di memoria o corrompere i dati contenuti nella memoria del sistema. In tal caso il controllo non sarebbe sicuro per lo scripting.

A conferma della serietà di questo argomento, è da notare che VBScript non include alcun metodo per accedere al Registro, per salvare file o creare oggetti.

Il controllo può anche essere contrassegnato come sicuro per l'inizializzazione *(safe for initialization)* per rassicurare gli utenti sul fatto che non esiste possibilità che un autore HTML possa danneggiare i loro computer fornendo dati non validi durante l'inizializzazione.

Utilizzando Package and Deployment Wizard per creare setup Internet, è possibile come già accennato, contrassegnare il controllo come sicuro per Pinizializzazione' In caso contrario Internet Explorer si rifiuterà di effettuare lo scaricamento del componente.

Anche contrassegnando il controllo come sicuro, l'impostazione predefinita di Explorer indica di rifiutare componenti ActiveX non firmati. Per modificare questa impostazione è sufficiente accedere alla scheda Security *(richiamabile dal menu* View) *e deselezionare l'impostazione* High Security. *È inoltre possibile impostare in maniera del tutto personalizzata la gestione degli aspetti di sicurezza utilizzando la finestra* Custom Settings *mostrata in Figura 27.12.*

Figura27.12 *Impostazione delle opzioni di sicurezza canalizzate.*

Utilizzo dei file creati da Package and Deployment Wizard

Dopo aver ottenuto tutte le informazioni di cui ha bisogno, Package and Deployment Wizard crea un'insieme di file, che comprende, solitamente:

- Un file .Cab, salvato nella posizione specificata, contenente il controllo. Nell'esempio il file è Confetti.Cab.
- Un file HTML di esempio contenente il tag `<OBJECT>` completo del riferimento CSLID per il controllo `Confetti`. Nel nostro caso il file si chiama Confetti. Htm.

- Una cartella di supporto che, nel nostro esempio, contiene i file forniti in input a Confetti.Cab. La cartella di supporto contiene il controllo, Confetti.Ocx, un file di informazioni per il setup, Confetti.Inf e un file di progetto per la creazione del file .Cab, chiamato Confetti.Ddf. Un ultimo file, Confetti.Bat, può essere utilizzato assieme al file di progetto per ricreare il pacchetto .Cab.

Inoltre, se il controllo ha bisogno di una licenza (come succede per molti dei controlli in commercio) sarà necessario creare un file LPK (License Package). Gli strumenti per creare un simile file possono essere scaricati dal sito http://www.microsoft.com/intdev/sdk/sdk.htm.

Utilizzo di un controllo ActiveX su Web

Ecco una porzione del contenuto del file Confetti.Htm generato da Package and Deployment Wizard:

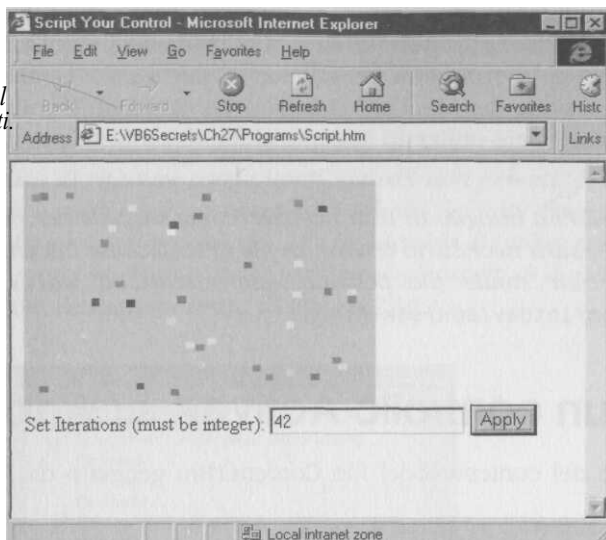
```
<HTML>
   OBJECT ID="Confetti" WIDTH=320 HEIGHT=240
      CLASSID="CLSID:2FDDA94E-5E8D-11D0-B8E9-0080C6026268"
      CODEBASE="confetti.CAB#version=1,0,0,0">
   </OBJECT>
</HTML>
```

Aprendo la pagina HTML con Explorer, ogni volta che l'evento Paint del controllo viene generato dal ridimensionamento della pagina in cui si trova, si ottiene la generazione di 5000 "confetti" attraverso l'iterazione predefinita.

Ogni controllo ActiveX personalizzato avrà un suo CSLID, che dovrebbe, teoricamente, essere unico nell'intero universo. Per evitare che vengano creati più CSLID per lo stesso controllo, assicuratevi di selezionare l'opzione Binary Compatibility *nella finestra* Project Properties *del progetto prima di compilare il controllo.*

Il comportamento del controllo ActiveX Confetti all'interno della pagina HTML di esempio è sostanzialmente privo di ogni utilità, se non addirittura noioso; infatti, prima di tutto, l'utente perde completamente il controllo di Internet Explorer mentre il controllo svolge il proprio lavoro, mentre, d'altra parte, lo sviluppatore potrebbe non desiderare che il controllo sparga i suoi "coriandoli" direttamente all'apertura della pagina (in seguito all'evento Resize), per permettere all'utente di decidere la quantità di coriandoli da generare (ovvero le iterazioni).

E possibile utilizzare applicazioni come FrontPage98, ActiveX Control Pad oppure il tradizionale Notepad per aggiungere gli oggetti appropriati, i valori delle proprietà e i comandi VBScript al sorgente. Il Listato 27.1, presente nel file Script.Htm nel CD-ROM, mostra come implementare una casella di input e un pulsante per manipolare il comportamento del controllo Confetti. *In questo modo l'utente sarà in grado di avviare e arrestare il controllo e impostare l'intervallo, come mostrato nella Figura 27.13.*

Figura27.13 *Esempio dipagina permanipolareil controlloConfetti.*

ActiveXControlPad è un'applicazione gratuita per manipolare componenti ActiveX su pagine Web che può essere scaricata dall'indirizzo http://www. micro-soft.com/workshop/author/cpad/download, htm.

Listato 27.1 *Scriptingper la manipolazione di un controtto ActiveX.*

```
<HTML>
<HEAD>
   <TITLE>Script Your Control</TITLE>
</HEAD>
<BODY>
<OBJECT ID="Confetti"
   CLASSID="CLSID:03E8A5EE-DA12 -11D1-B853-006008A093F0"
   CODEBASE="confetti.CAB"
   STYLE="TOP:17pt;LEFT:50pt;WIDTH:206pt;
      HEIGHT:131pt;TABINDEX:0;ZINDEX:0;">
        <PARAM NAME="_ExtentX" VALUE="7276">
        <PARAM NAME="_ExtentY" VALUE="4630">
        <PARAMNAME="Enabled"VALUE="0">
</OBJECT><BR>
Set Iterations (must be integer):
<Input type=text name=Iterations value=42>
<INPUT LANGUAGE="VBSCRIPT" Type=button Value=Apply
   ONCLICK="Confetti.Iterations = Iterations.value
   Confetti.enabled=True
   Confetti.refresh
   Confetti.enabled=False"
>
</BODY>
</HTML>
```

Un aspetto chiave dello script mostrato nel Listato 27.1 consiste nel fatto che si occupa di impostare il valore iniziale della proprietà personalizzata Enabled a False, utilizzando i parametri del tag <OBJECT>:

```
<PARAM NAME="Enabled" VALUE="0">
```

L'evento Paint del controllo Confetti non viene generato dalla pagina Web fintanto che l'utente non fa clic sul pulsante, causando l'invocazione del metodo Refresh del controllo e l'esecuzione del numero di iterazioni specificate.

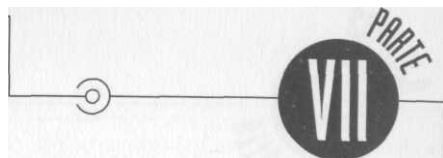
È da notare che, se il livello di sicurezza di Explorer fosse impostato a High, *si potrebbe avere l'impressione che la pagina sia stata caricata correttamente, ma il suo contenuto interattivo non sarebbe funzionante. Per caricare contenuti attivi con l'impostazione predefinita di sicurezza (*High), si *rende infatti necessario associare il controllo a un certificato di autenticità.*

In ogni caso il controllo Confetti non deve essere contrassegnato come sicuro per lo scripting, perché la proprietà Iterations del controllo è esposta e uno script potrebbe, in teoria, attingere alle risorse di sistema in modo incontrollabile e, se fossero disponibili più istanze del controllo, potrebbe addirittura utilizzare infinite iterazioni di numerosi controlli per bloccare completamente il sistema dell'utente.

Riepilogo

I controlli ActiveX creati con VB6 possono essere facilmente distribuiti attraverso appositi pacchetti. Package and Deployment Wizard, mettendo a disposizione numerose opzioni, è uno strumento che permette di raggiungere facilmente questo obiettivo. I controlli ActiveX e gli script che manipolano i loro membri esposti possono aggiungere contenuti eseguibili alle applicazioni Web. In questo capitolo:

- Avete imparato a utilizzare i controlli ActiveX e i browser.
- E stata introdotta la terminologia di base di Web.
- Avete scoperto come deve essere installato e configurato un controllo.
- E stato spiegato come installare un controllo da Web.
- Avete appreso importanti concetti riguardo i file cabinet (.Cab).
- Avete imparato a utilizzare il tag <OBJECT>.
- Avete imparato a utilizzare le coppie <PARAMETER NAME> per impostare i valori di inizializzazione per le proprietà personalizzate.
- Sono stati descritti i file generati da Package and Deployment Wizard.
- Avete imparato a contrassegnare come sicuri i controlli.
- È stato fornito un esempio di utilizzo di controlli da script in ambiente Web.

ESTENSIONE DELL'AMBIENTE

APPLICAZIONI INTERNET

- Aggiunta di capacità Web alle applicazioni VB
- Aggiunta di caratteristiche Internet ai controlli personalizzati
- Applicazioni basate sui documenti ActiveX
- Applicazioni DHTML
- Applicazioni Internet Information Server (IIS)

Nella Parte VI, abbiamo spiegato come usare VB6 per creare controlli ActiveX, i quali possono poi venire utilizzati da uno sviluppatore all'interno di VB o di altri ambienti di sviluppo. Nel capitolo finale della Parte VI si è visto come dispiegare i controlli VB sul Web. La Parte VII, tratta due argomenti: come si possono creare applicazioni in VB6 che usano il Web per andare oltre VB, e come si può alterare lo stesso ambiente VB.

Questo capitolo tratta alcune tecniche facili ed entusiasmanti che servono a estendere le capacità dei programmi VB6. La prima parte del capitolo spiega come aggiungere capacità Internet alle proprie applicazioni e ai propri controlli VB. Successivamente, spiegheremo come creare applicazioni basate sui documenti ActiveX, uno speciale tipo di programmi che comprende sia contenuto di documento che contenuto di programmazione che può venire "riprodotto" su Internet. Proseguiremo trattando la creazione di applicazioni basate su Dynamic HTML (DHTML) in VB6. Infine, tratteremo la creazione di applicazioni basate su Internet Information Server.

Aggiunta di capacità Web olle applicazioni Visual Basic

Le edizioni Professional e Enterprise di Visual Basic 6 comprendono due controlli ActiveX che servono a estendere le applicazioni VB con l'aggiunta di tecnologia relativa a Internet. Questi controlli sono:

- Il controllo WebBrowser, che incorpora gran parte della funzionalità di Internet Explorer e permette di aggiungere facilmente un browser Web a qualunque applicazione VB
- Il controllo Internet Transfer (trasferimento via Internet), che permette di trasferire facilmente dei file via Internet mediante numerosi protocolli

Il controllo WebBrowser

Per aggiungere il controllo WebBrowser alla Toolbox si seleziona *Microsoft Internet Controls* nella finestra di dialogo *Components,* come mostrato nella Figura 28.1.

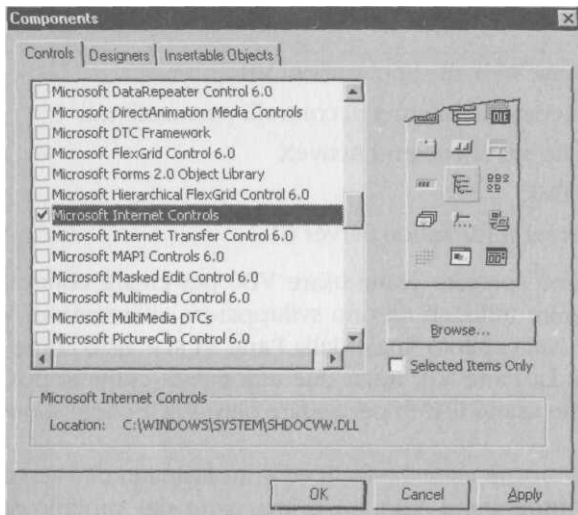
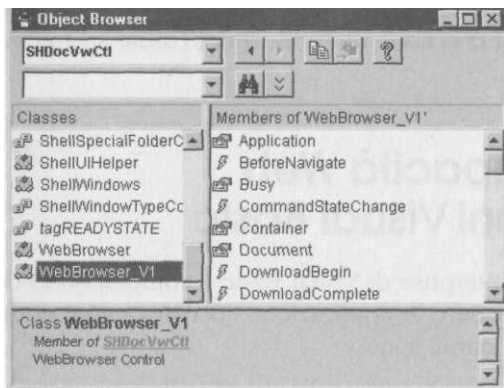
Figura 28.1 *Per usare il controllo WebBrowser nei propri progetti, si deve abilitare Microsoft Internet Controls nella finestra di dialogo Components.*

Il WebBrowser comparirà nell'Object Browser di VB come membro della libreria SHDocVwCtl, come mostrato nella Figura 28.2.

Figura 28.2 *Il controllo WebBrowser fa parte della libreria SHDocVwCtl.*

Dopo aver aggiunto il controllo WebBrowser alla Toolbox, lo si può porre in un form al solito modo. Con questo controllo, è molto facile aggiungere capacità di navigazione Web alle proprie applicazioni. Un progetto d'esempio che contiene il codice necessario è salvato col nome Browser.Vbp sul CD-ROM allegato.

Con l'Application Wizard di VB si genera un form contenente il controllo WebBrowser *che è molto simile a quello del progetto dimostrativo.*

Il file Browser.Frm del progetto d'esempio contiene, oltre al controllo WebBrowser, un Timer, una casella combinata (ComboBox), e un controllo Toolbar. Quando il form del browser viene caricato, il controllo WebBrowser deve venire ridimensionato e deve ricevere un indirizzo di partenza, come mostrato nel Listato 28.1:

Listato 28.1 *Dimensionamento e assegnazione di un indirizzo di partenza al WebBrowser.*

```
Public StartingAddress As String
Private Sub Form_Load()
   On Error Resume Next
   Me.Show
   tbToolBar.Refresh
   Form_Resize
   StartingAddress = "http://www.bearhome.com/cub/"
   cboAddress.Move 50, lblAddress.Top + lblAddress.Height + 15
   If Len(StartingAddress) > 0 Then
      cboAddress.Text = StartingAddress
      cboAddress.AddItem cboAddress.Text
      'try to navigate to the starting address
      timTimer.Enabled = True
      brwWebBrowser.Navigate StartingAddress
   End If
End Sub
```

Il codice di ridimensionamento del form gestisce il ridimensionamento del WebBrowser:

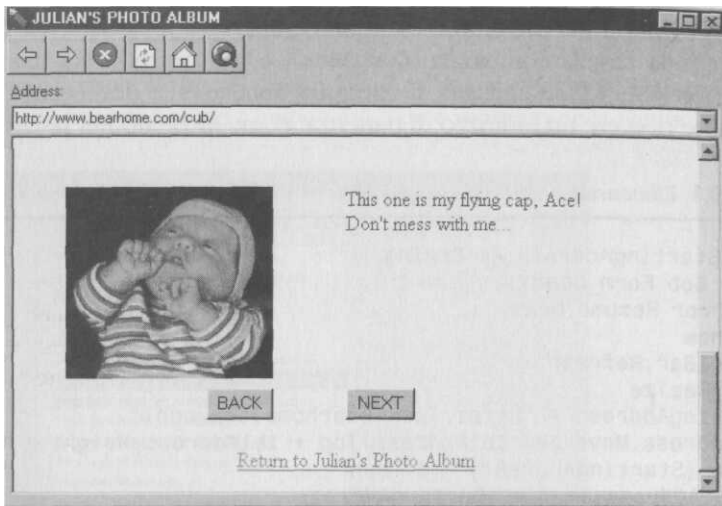
```
Private Sub Form_Resize()
   cboAddress.Width = Me.ScaleWidth - 100
   brwWebBrowser.Width = Me.ScaleWidth - 100
   brwWebBrowser.Height = Me.ScaleHeight - _
      (picAddress.Top + picAddress.Height) - 100
End Sub
```

E compito del controllo Timer, dopo che è stato abilitato, continuare a provare facendo scattare il suo evento Timer a brevi intervalli fino a che il WebBrowser si connette all'indirizzo specificato:

```
Private Sub timTimer_Timer()
   If brwWebBrowser.Busy = False Then
      timTimer.Enabled = False
      Me.Caption = brwWebBrowser.LocationName
   Else
      Me.Caption = "Sto lavorando..."
```

```
    End If
End Sub
```

Supponendo di avere una connessione Internet attiva e un indirizzo valido tutt ciò dovrebbe funzionare tranquillamente. La Figura 28.3 mostra il controllo Web Browser nel progetto d'esempio con una pagina Web caricata.

Figura 28.3 *È facile usare il controllo WebBrowser per connettere il proprio progetto al Web.*

Dopo che l'Application Wizard di VB ha preparato un browser, i pulsanti detta sua toolbar invocano i metodi dell'oggetto WebBrowser, *quando ricevono un clic. Per la maggior pane, questi funzionano bene. Tuttavia, il pulsante* Home, *che chiama il metodo* GoHome, *e il pulsante* Search, *che invoca il metodo* GoSearch, *sono "cablati" nelle impostazioni di Microsoft Internet Explorer.*

Questo ha più senso di quanto non appaia a prima vista, perché il controllo WebBrowser è essenzialmente un'interfaccia fra le applicazioni e la libreria di automazione dell'oggetto Internet Explorer. L'effetto risultante, comunque, è che se si vuole controllare l'effetto dei comandi *Home* o *Search,* non si possono usare i metodi incorporati. Per farlo, si dovrebbero sostituire i metodi GoHome o GoSearch del controllo con chiamate al metodo Navigate in cui si indicano i propri URL.
Il Listato 28.2 mostra il codice di Toolbar revisionato con una posizione "personalizzata" come pagina iniziale:

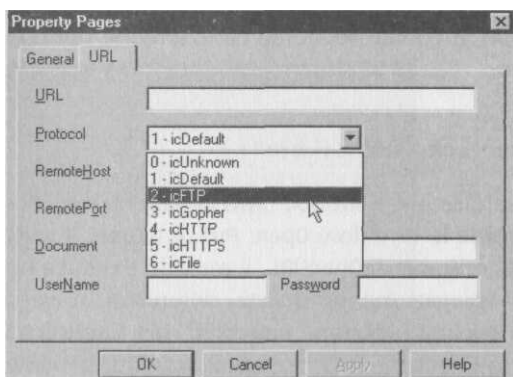
Listato 28.2 *Aggiunta di un indirizzo personalizzato per la pagina iniziale.*

```
Private Sub tbToolBar_ButtonClick(ByVal Button As Button)
   On Error Resume Next
   timTimer.Enabled = True
   Select Case Button.Key
      Case "Back"
         brwWebBrowser.GoBack
      Case "Forward"
```

```
            brwWebBrowser.GoForward
        Case "Refresh"
            brwWebBrowser.Refresh
        Case "Home"
            brwWebBrowser.Navigate StartingAddress 'GoHome
        Case "Search"
            brwWebBrowser.GoSearch
        Case "Stop"
            timTimer.Enabled = False
            brwWebBrowser.Stop
            Me.Caption = brwWebBrowser.LocationName
    End Select
End Sub
```

Il controllo Internet Transfer

Per aggiungere il controllo Internet Transfer alla Toolbox dalla finestra di dialogo *Components,* si deve impostare l'opzione *Microsoft Internet Transfer Control 6.0.* Il nome di classe di questo controllo, come si può vedere nella finestra *Properties,* è Inet. Nell'Object Browser si presenta come `InetCtlsObjects.Inet`. Lo scopo di Inet è di facilitare alle applicazioni l'esecuzione di trasferimenti di file via Internet usando i comuni protocolli di Internet. La Figura 28.4 mostra la finestra di dialogo *Custom Property Page* per un esemplare del controllo. Si seleziona un protocollo dai tipi enumerativi mostrati nella casella di riepilogo a discesa *Protocol.*

Figura 28.4 *Il controllo Internet Transfer serve a implementare trasferimenti di file mediante protocolli standard di Internet.*

Tra gli usi del controllo Internet Transfer, c'è l'aggiunta di un client FTP all'applicazione, l'automazione degli scaricamenti *(download)* da un sito FTP pubblico, e lo scaricamento di porzioni di siti Web. Altri usi di questo controllo flessibile, potente e facile da usare comprendono, come vedremo tra un attimo, l'esecuzione di programmi CGI (Common Gateway Interface) su server Web remoti. Il funzionamento di base di tale controllo solitamente comprende quattro passi:

1. Impostare la proprietà AccessType del controllo. Questo riguarda il tipo di accesso a Internet usato dal controllo. Di default, il controllo userà le impostazioni d'accesso a Internet trovate nel Registro di configurazione. La Tabella 28.1 mostra le possibili impostazioni.

Tabella 28.1 *Impostazioni detta proprietà AccessType per il controllo Internet Transfer.*

Costante	Valore	Descrizione
icUseDefault	0	Usa le impostazioni d'accesso di default trovate nel registro di configurazione del sistema
icDirect	1	Il controllo ha una connessione diretta a Internet
icNamedProxy	2	Istruisce il controllo a usare il server proxy specificato nella proprietà Proxy

2. Invocare il metodo OpenURL del controllo con un URL valido. Questo passo non è necessario se si vuole solo eseguire un comando sul server remoto.
3. Usare il metodo Execute del controllo con un URL valido e con un comando che funziona con il protocollo in uso.
4. Usare il metodo GetChunk del controllo per recuperare i dati inviati dal server remoto al buffer.

Generalmente, si può lasciare il protocollo impostato a icDefault, *e lasciare che il controllo determini quale protocollo usare a seconda della risposta del server.*

Dopo aver impostato la proprietà AccessType, si può usare il metodo OpenURL per ottenere un file. Per esempio, il codice seguente caricherebbe del codice sorgente HTML in una casella di testo:

```
Text1.Text = Inet1.OpenURL _
   ("http://www.theserver.com/default.htm")
```

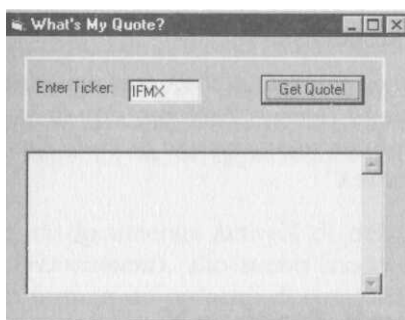
Se si volesse salvare il file su disco, si potrebbe farlo facilmente con i comandi standard per manipolare i file come le istruzioni Open, Put, e dose. Il metodo OpenURL opera in modo sincrono. Al contrario di OpenURL, il metodo Execute opera in modo asincrono. Per avere una spiegazione di questa differenza si veda la sezione "Comunicazione asincrona e comunicazione sincrona" nel Capitolo 20. Ciò comporta che se si usa il metodo Execute per recuperare dei dati da un server, si deve tenere traccia dello stato di connessione del controllo mediante il suo evento StateChanged. Altrimenti, non si saprà mai quando ha finito. Per salvare o recuperare i dati posti nel buffer del controllo, si può porre nell'evento StateChanged una chiamata al metodo GetChunk.
Il metodo Execute prende quattro parametri: l'URL, l'operazione, i dati, e le intestazioni della richiesta. Le operazioni FTP normalmente ometteranno gli ultimi due, che sono facoltativi. Si può usare l'argomento operazione per eseguire la maggior parte delle operazioni FTP standard come ricevere dati, inviare dati, e creare directory.

ci può usare il metodo Execute con il protocollo HTTP per usare i comandi standard del protocollo per richiedere dati da un server Web. Come molti sapranno, questi comandi sono GET, HEAD, POST, e PUT.

L'esempio salvato sul CD-ROM con nome GetQuote.Vbp usa il controllo Internet Transfer e il server di quotazioni in borsa (quote server) Yahoo per restituire l'HTML contenente la quotazione corrente in borsa per qualunque azione.

Recuperare informazioni dal server di quotazioni in borsa Yahoo non è un esempio particolarmente utile nel mondo reale. Ci sono molte entusiasmanti applicazioni pratiche per sfruttare tecnica. Si può pensarla essenzialmente come un modo per eseguire qualsiasi comando HTTP, su qualunque server Web, dall'interno di un programma Visual Basic.

La Figura 28.5 mostra l'applicazione che restituirà il codice HTML comprendente la quotazione corrente in borsa per il titolo InForm1x Software, avente codice IFMX, prelevandolo dal server Web Yahoo. Di default l'applicazione visualizzerà l'informazione relativa al codice MSFT, cioè al titolo Microsoft, ma si può introdurre qualunque codice si desideri. Naturalmente, affinchè l'applicazione funzioni, si deve essere connessi a Internet.
Per rendere utile nel mondo reale questa visualizzazione, si vorrà o visualizzarla in formato HTML, come fa Yahoo sul suo sito, o estrarre il sorgente HTML della quotazione e scartare il resto. Oltre alla quotazione effettiva, all'ora, e a informazioni sulle oscillazioni del titolo, il server di quotazioni in borsa Yahoo restituisce parecchio altro codice HTML; si tratta di tutte quelle cose che si vedono quando si visita il sito http://quote.yahoo.com, come gli striscioni *(banner)* pubblicitari.

Figura 28.5
// metodo Execute del controllo Internet Transfer serve a lanciare comandi HTTP su un server Web.

Ecco il codice che manda un comando HTTP POSI con un codice di titolo azionario al server di quotazioni in borsa Yahoo:

```
Private Sub Gommane!1_Click()
   Dim strURL As String, strFormData As String
   strURL = "http://quote.yahoo.eom/q"
   strFormData = "symbol=" & Trim(ticker.Text)
   Inetl.Execute strURL, "POSI", strFormData
End Sub
```

Ecco il codice che trasferisce alla casella di testo il contenuto del buffer del controllo, dopo che il server ha completato la risposta al comando POSI:

```
Private Sub Inet1_StateChanged(ByVal State As Integer)
    Dim vtData As Variant ' Variabile dei dati.
    Select Case State
    ' ... Altri casi non riportati
    Case icResponseCompleted ' 12
    vtData = Inetl.GetChunk(4072, icString)
    Text1 = vtData
    End Select
End Sub
```

Si noti che il primo argomento del metodo GetChunk *specifica, in numero di byte quanto codice HTML il server deve restituire. Se non si rende abbastanza grande questo numero, si potrebbe non ricevere tutto il codice HTML che si desidera.*

Aggiunta di caratteristiche Internet ai controlli

Un modo importante di aggiungere capacità Internet alle proprie applicazioni VB6 è di scrivere le funzioni relative nei controlli che si creano. Per ulteriori informazioni sulla creazione di controlli, si veda la Parte VI. I controlli ActiveX creati usando Visual Basic 6 possono supportare una varietà di caratteristiche relative a Internet. In generale, queste caratteristiche richiedono per funzionare che il contenitore del controllo sia Internet Explorer. Per esempio, gli oggetti UserControl possono supportare lo scaricamento asincrono di valori di proprietà, come le proprietà Picture che possono contenere mappe di bit. Attraverso la proprietà Hyperlink dell'oggetto controllo d'utente, si può anche richiedere che un browser salti a un URL, o che navighi attraverso la sua cronologia.

I *valori di proprietà per i componenti e i controlli ActiveX possono adesso essere fatti persistere in un* PropertyBag *globale in Internet Explorer, dando la possibilità di salvare le impostazioni e i dati quando un utente naviga fuori da una pagina contenente un controllo o documento ActiveX.*

Le applicazioni basate sui documenti ActiveX

I documenti ActiveX sono un nuovo tipo di applicazione che si può creare in Visual Basic. I progetti di documenti ActiveX servono a creare applicazioni complesse che sono interamente contenute in un'applicazione host, come Internet Explorer.

Le applicazioni basate su documenti ActiveX sono composte da componenti ActiveX server di automazione e da "documenti". Questi documenti chiamano il server di automazione che sta dietro l'applicazione basata su documento ActiveX esattamente nello stesso modo in cui i documenti di Word chiamano gli oggetti di automazione esposti nel server di documenti Word, Winword.Exe.

I file di documento (.Vbd) creati quando si compila un'applicazione basata su documenti ActiveX usano la memoria *(Storage)* strutturata OLE, così che i dati del documento possono venire acceduti e manipolati per mezzo delle interfacce OLE standard, proprio come succede ai documenti di Word e di Excel. In altre parole, un'applicazione basata su documenti ActiveX è composta da due parti concettualmente distinte: i componenti ActiveX che funzionano come server di automazione OLE, e i documenti che sono progettati per interagire con il server.
Compilando un'applicazione Visual Basic basata su documenti, si crea sia un documento (un file .Vbd) che un corrispondente server ActiveX (un file .Dll o .Exe). Il file .Vbd sta al server ActiveX come un file .Doc sta a Winword.Exe.
Le applicazioni basate su documenti ActiveX hanno bisogno di un'applicazione host per venire eseguite, proprio come i controlli ActiveX hanno bisogno di un contenitore. Tra le applicazioni che ospitano documenti ActiveX ci sono Microsoft Internet Explorer, il Raccoglitore Office (Microsoft Office Binder), e l'IDE di Visual Basic.

Si può usare la funzione CreateToolWindow per *creare una finestra per uno strumento ancorabile nell'IDE di Visual Basic, che potrebbe contenere un'applicazione basata su documenti ActiveX, come, per esempio, un editor di risorse potenziato.*

Creazione di un'applicazione basata su documenti ActiveX

Per creare una nuova applicazione basata su documenti ActiveX, si seleziona *ActiveX Document EXE* o *ActiveX Document DLL* dalla finestra di dialogo *New Project.* Selezionando un progetto EXE si ottiene un server di automazione *out-of-process,* mentre selezionando un progetto DLL si ottiene un server *in-process.*

Le applicazioni basate su documenti compilate come DLL probabilmente verranno eseguite molto più velocemente di quelle compilate come EXE, ma sono soggette a più vincoli. Per esempio, in un'applicazione DLL non si può aprire un form non modale.

Un progetto di documento ActiveX di default contiene un oggetto documento d'utente *(UserDocument),* allo stesso modo in cui un progetto Standard EXE è basato su un form, e un progetto di controllo è basato su un controllo d'utente. I controlli sono posti sul documento d'utente, mentre i moduli e il codice sono aggiunti al progetto per adempiere alla funzionalità desiderata. Quando viene compilato il progetto di documento ActiveX, oltre al file contenente il server di automazione, viene creato un file .Vbd per ogni documento d'utente dell'applicazione.

Molte delle tecniche implicate nella creazione di applicazioni basate su documenti ActiveX sono affini a quelle usate nella creazione di controlli ActiveX. Si veda la Parte VI per ulteriori informazioni. Bisogna conoscere anche le tecniche di automazione ActiveX, trattate nel Capitolo 22 e nel Capitolo 23.

Conversione di applicazioni esistenti

Concettualmente, le applicazioni basate su documenti ActiveX sono più vicine alle applicazioni basate su controlli ActiveX che ai normali progetti. Per dirne una, sia i documenti ActiveX che i controlli ActiveX devono funzionare entro un contenitore o host. Comunque, se si vuole convertire un progetto standard esistente, ci sono alcuni buoni candidati. Si noti che le applicazioni basate su documenti ActiveX non possono includere il controllo contenitore OLE. Il wizard per la migrazione di documenti ActiveX (ActiveX Document Migration Wizard) è un'aggiunta di Visual Basic che aiuta a convertire progetti standard in progetti di documenti ActiveX.

I file .Vbd

Dopo che un'applicazione basata su documenti ActiveX è stata compilata, i file .Vbd creati dalla compilazione vengono aperti da un'applicazione host, come Internet Explorer. Ogni file .Vbd contiene un riferimento all'identificatore di classe del suo server di automazione. È utilizzato anche dall'applicazione per immagazzinare dati persistenti relativi al documento.

Dopo che il file . Vbd è stato generato, si po' ridenominare l'estensione come si desidera. Per esempio, se ActXDoc. Vbd fosse ridenominato in ActXDoc.Bad, sarebbe ancora perfettamente funzionale.

L'implementazione dei documenti ActiveX

La Figura 28.6 mostra un esempio di applicazione basata su documenti ActiveX aperta in Internet Explorer. Quando l'utente fa clic sul pulsante *NavigateTo,* Internet Explorer apre l'URL specificato, come mostrato nella Figura 28.7.

Figura 28.6 *Le applicazioni basate su documenti ActiveX possono venire ospitate da Internet Explorer.*

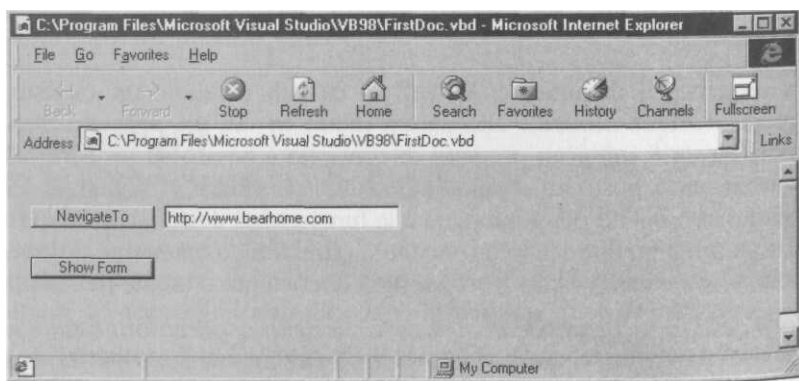
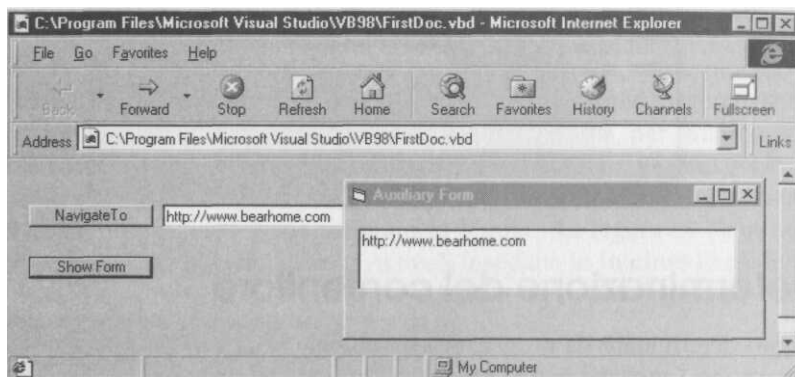



Figura 28.7
Le applicazioni basate su documenti ActiveX supportano i metodi di navigazione di Explorer.

Si può anche fare in modo che il documento ActiveX, mentre è ospitato da Internet Explorer, mostri un form standard creato in Visual Basic (Figura 28.8).

Figura 28.8
Le applicazioni basate su documenti ActiveX possono visualizzare form standard creati in VB.

Supponendo che l'host per l'applicazione basata su documenti ActiveX sia Internet Explorer, si può usare il metodo Navigatelo *di un oggetto* Hyperlink *del documento d'utente per passare da un documento a un altro. Si dovrebbero usare gli URL formattati al solito modo. Per esempio, per andare a un file locale:*

```
UserDocument.Hyperlink.NavigateTo "file://C:\activeX\Second.Vbd"
```

La stessa sintassi può venire usata per saltare a una posizione sul Web:

```
UserDocument.Hyperlink.NavigateTo "http://www.bearhome.com"
```

Nel programma d'esempio, l'argomento per il metodo Navigatelo dell'oggetto è il contenuto di una casella di testo:

```
UserDocument.Hyperlink.NavigateTo txtURL.Text
```

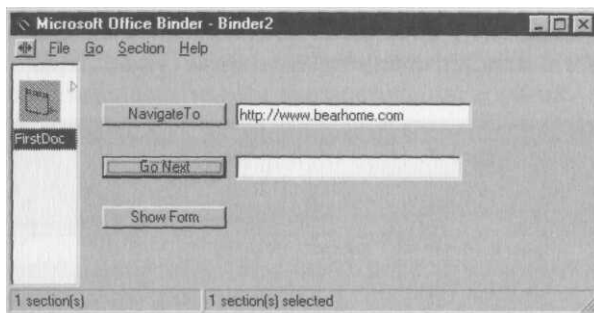
Ecco come aprire il form VB ausiliario:

```
PrivateSubcmdShowForm_Click()
   ' Visualizza il form ausiliario e imposta
```

```
   ' le proprietà text di txtAux all'URL di FirstDoc.
   frmAux.txtAux.Text = txtURL.Text
   frmAux.Show vbModal
End Sub
```

Documenti ActiveX e il Raccoglitore Office

I documenti ActiveX possono anche venire aggiunti al Raccoglitore Office (Micro soft Office Binder), come mostrato in Figura 28.9. Per aggiungere un documento ActiveX a un Raccoglitore, scegliere *Add from File* (aggiungi da file) dal menu *Section* (sezione) del Raccoglitore. Selezionare l'appropriato file di documento ActiveX. A meno che l'estensione sia stata cambiata, questo sarà un file .Vbd.

Figura 28.9 *Si possono aggiungere documenti ActiveX a un Raccoglitore.*

Determinazione del contenitore

Si può determinare da programma il contenitore di un documento ActiveX usando l'istruzione `TypeName` con la proprietà `Parent` del documento d'utente:

```
Dim strWhat As String
StrWhat = TypeName (UserDocument.Parent)
```

La Tabella 28.2 mostra tre possibili stringhe restituite. Si noti che queste stringhe *distinguono* tra maiuscolo e minuscolo!

Tabella 28.2 *Stringhe rese dal contenitore.*

Contenitore	Stringa
Raccoglitore (Binder)	Section
Explorer	IwebBrowser2
Finestra di strumento dell'IDE di VB6	Window

Il progetto d'esempio di documento ActiveX salvato col nome Contain. Vbp sul CD-ROM determina se il suo contenitore è Internet Explorer. Se no, visualizza la stringa costante del contenitore, come mostrato nel Listato 28.3.

listato 28.3 *Determinazione da programma di un host di documento ActiveX.*

```
Private flgShow As Boolean 'Module level

Private Sub UserDocument_Show()
   If Not flgShow Then
      Dim strContainer As String
      strContainer = TypeName(UserDocument.Parent)
      Select Case strContainer
         Case "IWebBrowser2"
            'Contenitore supportato, non ci sono problemi
            MsgBox "Internet Explorer"
         Case Else
            MsgBox "Per favore, usa Internet Explorer!"
            MsgBox strContainer
      End Select
   flgShow = True
   End If
End Sub
```

Questo codice va nell'evento Show del documento d'utente, che scatta quando il documento ActiveX viene posizionato in un contenitore. Siccome l'evento Show scatta ancora ogni volta che il documento ActiveX viene mostrato, per evitare risultati erronei, alla routine è stato aggiunto un flag a livello di modulo. Questo assicura che il contenitore sia verificato solamente quando il documento ActiveX è posizionato la prima volta, non quando viene mostrato in seguito. La Figura 28.10 mostra l'applicazione Contain basata su documenti ActiveX insediata in Internet Explorer.

Figura 28.10 *Si può determinare da programma quale applicazione stia ospitando il documento ActiveX.*

Applicazioni basate su DHTML

Le applicazioni basate su Dynamic HTML (DHTML) vengono implementate usando estensioni ad HTML con cui si può posizionare con precisione e associare script a tutti gli elementi delle pagine Web. Questo è un concetto estremamente potente perché il browser Web che capisce il DHTML è diventato l'interprete che elabora il codice. Dovrebbe funzionare su una piattaforma sulla quale funziona il browser. Inoltre, DHTML permette di liberare il server di una gran quantità di elaborazione.

Siccome una maggior quantità di lavoro viene fatta da client, cioè dal browser Web, il server è alleggerito da una parte del carico di elaborazioni pesanti, e c'è meno bisogno di fare mólti tira e molla attraverso la rete.
Se tutto ciò sembra troppo bello per essere vero, c'è, come si può sospettare uno scotto da pagare. Per dirne una, non tutti i browser capiscono il DHTML. E come se non bastasse, Netscape Communicator 4.0 e Microsoft Internet Explorer 4.0, parlano dialetti diversi.

Il DHTML generato da VB6 è destinato a venire usato solamente in Internet Explorer 4.01 o successivi.

Determinazione del browser

Questo lascia alcune opzioni: si può dire agli utenti che le proprie applicazioni basate su DHTML funzionano solamente con un browser specifico, come Internet Explorer; si può scrivere un'applicazione che verifica quale browser viene usato e produce del codice DHTML appropriato sia per Fxplorer che per Navigator; oppure si può verificare la presenza di un browser DHTML e dirigerlo alla pagina scritta appositamente.

Il Listato 28.4, salvato sul CD-ROM col nome Dynamic.Htm, mostra come usare JavaScriptper implementare la rilevazione della versione di browser e il reindirizzamento automatico.

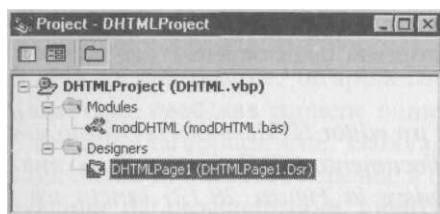
Listato 28.4 *Rilevazione della versione del browser e reindirìzzamento.*

```
<HTML>
<HEAD>
<SCRIPT Language="JavaScript">
if (navigator.userAgent.indexOf("Mozilla/4.0") != -1) {
  // In esecuzione un browser versione 4
  if (navigator.appName == "Netscape") {
     //Netscape 4
     window.location.href = "/dynamic/n4begin.htm"
     } else {
     //Explorer 4
     window.location.href="/dynamic/ie4begin.htm"
     }
  } else {
  //Browser versione 3
     window.location.href="/dynamic/v3begin.htm"
}
</SCRIPT>
<TITLE>Dynamic Start
</TITLE>
</HEAD>
<BODY>
<center>
<H1>
   Se leggete questa scritta,
```

```
il vostro browser non supporta JavaScript!
</H1>
</BODY>
</HTML>
```

DHTML e VB6

per creare un'applicazione basata su DHTML per Internet Explorer, si apra un nuovo progetto e si selezioni *DHTML Application.* Come mostrato in Figura 28.11, VB6 creerà una struttura di progetto contenente due moduli. Il primo è un modulo di codice standard, modDHTML.Bas, che contiene le routine necessarie a usare il propertyBag di Internet Explorer per far persistere informazioni passando da una pagina DHTML a un'altra. Il secondo è un designer di pagina DHTML.

Figura 28.11
Il progetto DHTML di default inizia con due moduli.

Si può anche aggiungere un designer DHTML a un normale progetto VB selezionando Add DHTML Page *dal menu* Project, *purché sia stata impostata l'opzione* DHTML Page *nella scheda* Designers *dellafinestra di dialogo* Components.

Il designer di pagina DHTML è diviso in due pannelli come mostrato nella Figura 28.12. Il pannello a destra viene usato per aggiungere controlli alla pagina, mentre il pannello a sinistra viene usato per impostare le proprietà degli oggetti e per aggiungere procedure agli oggetti.

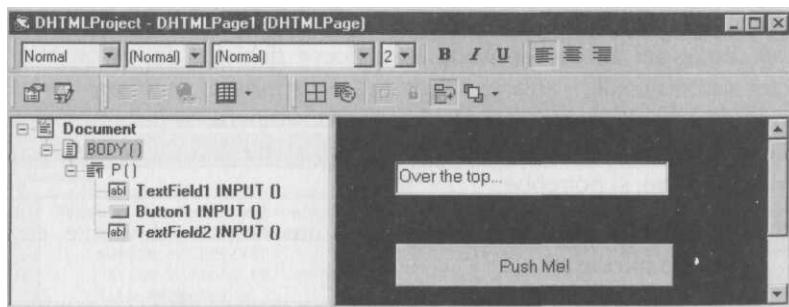
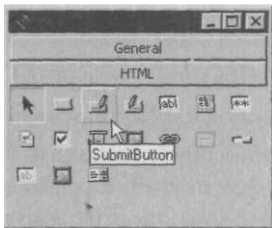
Figura 28.12
Un designer di pagina DHTML pone in grado di manipolare facilmente la gerarchia di oggetti DHTML

Quando il progetto DHTML è stato creato, i controlli HTML intrinseci sono aggiunti al loro pannello della Toolbox, come si vede nella Figura 28.13. Tutti altri controlli sono stati disabilitati.

Figura 28.13
Unprogetto DHTML fornisce automaticamente controlli HTML intrinseci.

Questi controlli intrinseci corrispondono agli oggetti dei form HTML standard, a cui molti saranno abituati: pulsanti di invio, caselle di testo, e così via. Dopo che sono stati aggiunti i controlli al form, si possono usare la finestra *Properties* e l'editor di codice di VB per modificare le proprietà degli oggetti e aggiungere procedure di eventi.

È opportuno notare che VB non è un editor HTML. Molti vorranno usare anche un editor esterno per ottenere un collocamento HTML preciso. Il secondo pulsante da sinistra sul designer DHTML (vedere la Figura 28.12) lancia un editor HTML esterno.

Applicazioni basate su Internet Information Server (IIS)

Un'applicazione basata su IIS è un'applicazione Visual Basic che usa una combinazione di codice HTML e di codice Visual Basic compilato in un'applicazione dinamica, basata su browser. Queste applicazioni devono venire eseguite in collaborazione con il server Web Microsoft Internet Information Server (IIS). Un'applicazione basata su IIS viene dispiegata sul server Web, dove riceve richieste da un browser, esegue del codice associato alle richieste, e restituisce risposte al browser, normalmente sotto forma di pagine HTML. Nella sua forma più semplice, si può usare un'applicazione basata su IIS per intercettare una richiesta di utente e restituire una pagina HTML al browser. Inoltre, si potrebbe:

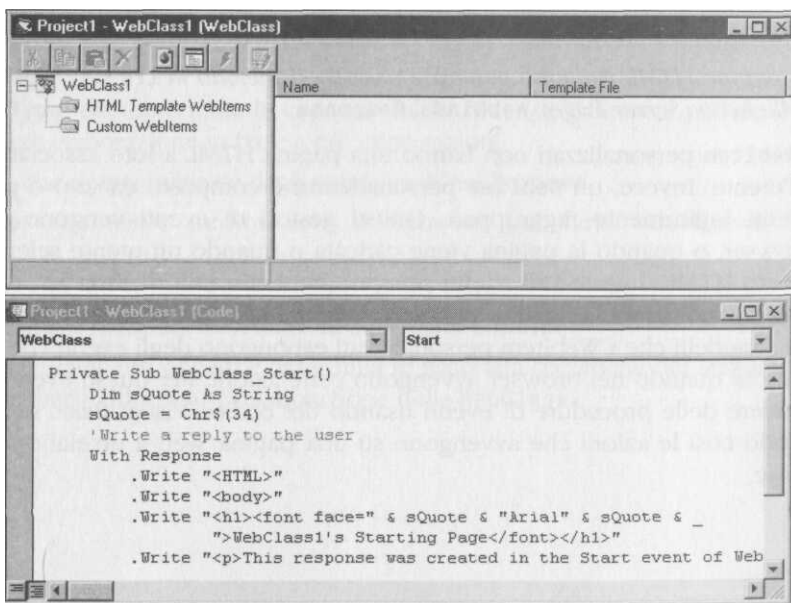
- Manipolare dei database in risposta a una richiesta di utente, scrivendo o leggendo informazioni.
- Recuperare pagine HTML e sostituire porzioni di esse con contenuto dinamico prima di mandarle al browser.
- Creare dinamicamente elementi HTML e generare eventi per essi al volo, in fase di esecuzione.

Applicazioni basate su IIS, DHTML e ASP

Al contrario delle applicazioni DHTML, quelle basate su IIS sono basate su server, invece che basate su client. Funzionano solamente con il server Web IIS, ma, diversamente da DHTML, non sono limitate a una specifica implementazione di browser. Come le applicazioni basate su IIS, anche le applicazioni basate su Active Server pages (ASP, pagine attive di server) risiedono sul server. Le Active Server Pages sono pensate per combinare lo scripting di lato server con il codice HTML. Le applicazioni basate su IIS servono agli sviluppatori Visual Basic per costruire applicazioni basate sul Web, invece che pagine Web. Le applicazioni basate su IIS permettono complesse elaborazioni di dati commerciali e facile accesso da quasi ogni piattaforma e browser.

Gli oggetti WebClass

All'utente, un'applicazione basata su IIS appare costituita da una normale serie di pagine HTML. Per lo sviluppatore, un'applicazione basata su IIS è costituita da oggetti WebClass. Ogni WebClass contiene numerosi WebItem. La WebClass funge da "cervello" centrale dell'applicazione, elaborando dati dal browser e servendo informazioni agli utenti. La WebClass risponde a queste richieste in base a una serie di procedure definite dal programmatore. I WebItem sono pagine HTML e altri dati che la WebClass può mandare al browser in risposta a una richiesta.
La Figura 28.14 mostra il designer degli oggetti WebClass visualizzato quando si apre una nuova applicazione basata su IIS in VB6.

Figura 28.14
VB6 fornisce un designer per gli oggetti WebClass quando si apre una nuova applicazione basata su IIS.

Una WebClass è un componente ActiveX di Visual Basic che risiede su un serve Web e risponde all'input proveniente dal browser. Quando si crea un'applicazione basata su IIS, si creano le sue WebClass usando il designer delle WebClass. Una WebClass è associata per tutta la sua vita a uno e un solo client. Visual Basic crea un esemplare logico della WebClass per ogni client che vi accede. Comunque, per ogni client, la WebClass è in grado di mantenere lo stato corrente fra le richieste.
Ogni WebClass di un'applicazione basata su IIS è associata a un file Active Server Pages (ASP) generato automaticamente da Visual Basic quando si compila o si esegue il progetto. Il file ASP ospita la WebClass sul server Web. Inoltre, genera il componente per la fase di esecuzione della WebClass quando l'applicazione viene avviata la prima volta, e lancia il primo evento della vita della WebClass.

I WebItem

Una WebClass contiene dei WebItem che usa per fornire il contenuto al browser e per esporre eventi per l'elaborazione. Un WebItem può essere una delle due seguenti cose:

- Un file modello (template) HTML
- Un WebItem personalizzato

I file modello HTML sono pagine HTML che vengono associate alla propria WebClass. Quando la WebClass riceve una richiesta, può servire le pagine HTML al browser per visualizzarle. I modelli differiscono dalle pagine HTML regolari solamente in quanto contengono spesso aree di sostituzione che la WebClass può elaborare prima di mandare la pagina al browser. Questo permette di personalizzare le risposte.

Le pagine HTML vengono generate usando il metodo Write *dell'oggetto* Response *delle Active Server Pages,* WebClass.Response.

I WebItem personalizzati non hanno una pagina HTML a loro associata da rendere all'utente. Invece, un WebItem personalizzato è composto da uno o più gestori di eventi logicamente raggruppati. Questi gestori di eventi vengono chiamati dal browser, o quando la pagina viene caricata o quando un utente seleziona un elemento HTML. I gestori di eventi possono generare una risposta al browser o passare l'elaborazione a un altro dei WebItem della WebClass.
Sia i modelli che i WebItem personalizzati espongono degli eventi che la WebClass elabora quando nel browser avvengono certe azioni. Per questi eventi, si possono scrivere delle procedure di eventi usando del codice Visual Basic standard, collegando così le azioni che avvengono su una pagina Web a un'elaborazione Visual Basic.

Struttura di un'applicazione basata su MS

Un'applicazione basata su IIS composta dai seguenti pezzi. Molti di questi sono generati automaticamente per i programmatori quando si costruisce un progetto IIS in VB6. Tra questi pezzi ci sono:

- Le WebClass
- I modelli HTML
- I WebItem personalizzati
- Un file ASP usato per ospitare la WebClass nell'Interne! Information Server. L'ASP viene generato automaticamente quando si crea un progetto WebClass.
- Un componente WebClass per la fase di esecuzione, Mswcrun.Dll, usato per elaborare le richieste
- Un progetto DLL che contiene il proprio codice Visual Basic e a cui si accede per mezzo di Mswcrun.Dll

Il modello di oggetti delle applicazioni basate su IIS

Le applicazioni basate su IIS sono ospitate da un file ASP e fanno uso di vari oggetti del modello di oggetti delle Active Server Pages per accedere e manipolare informazioni su una pagina HTML. Gli oggetti ASP che una WebClass può usare comprendono:

- Application, che gestisce lo stato corrente che è condiviso fra esemplari multipli di WebClass
- BrowserType, che determina le capacità del browser dell'utente e prende decisioni di elaborazione in base a tali informazioni
- Request, che riceve richieste dagli utenti finali nel browser
- Response, che serve informazioni al browser per farle visualizzare all'utente
- Session, che mantiene informazioni sulla corrente sessione di utente, immagazzina e recupera informazioni sullo stato corrente
- Server, che crea altri oggetti e determina le proprietà specifiche del server che potrebbero influenzare l'elaborazione della WebClass

Riepilogo

Questo capitolo ha spiegato come aggiungere caratteristiche Internet alle applicazioni mediante il controllo WebBrowser; a gestire i trasferimenti di file con il contrailo Internet Transfer; a creare applicazioni basate su documenti ActiveX; a usare VB6 per creare applicazioni basate su DHTML; e i fondamenti dell'utilizzo di VB6 per creare applicazioni basate su Internet Information Server (IIS).

- È stato spiegato l'uso del controllo WebBrowser per aggiungere capacità Web alle proprie applicazioni.
- Abbiamo compreso qualcosa sul funzionamento del controllo Internet Transfer.
- Abbiamo visto l'utilizzo del controllo Internet Transfer per eseguire comandi HTTP su un server Web remoto.
- Abbiamo compreso qualcosa sulle applicazioni basate sui documenti ActiveX.
- Abbiamo trattato il file .Vbd.
- Abbiamo scoperto come vengono "ospitati" i documenti ActiveX.
- Abbiamo visto come aggiungere un documento ActiveX al Raccoglitore Office.
- Abbiamo scoperto come determinare da programma un contenitore di documento ActiveX.
- Abbiamo visto che cos'è DHTML.
- È stato spiegato come determinare quale browser stia visualizzando una pagina e come redirigerlo appropriatamente.
- Abbiamo visto l'utilizzo di VB6 per creare applicazioni basate su DHTML.
- Abbiamo introdotto le applicazioni basate su Internet Information Server (IIS).

CREAZIONE DI UN ADD-IN PER VISUAL BASIC

- Che cos'è un add-in
- Concetti relativi all'oggetto VBIDE
- Come creare un add-in personalizzato
- Creazione di un add-in più complesso per modificare i colori, che interagisce con i progetti VB

Gli add-in (aggiunte) di Visual Basic sono componenti server ActiveX che interagiscono con istanze dell'ambiente di sviluppo Visual Basic. In generale, lo scopo di un add-in è facilitare un'operazione di sviluppo complessa o noiosa. Perciò, gli "utenti" degli add-in sono probabilmente degli sviluppatori. Per questi utenti, un • add-in si presenta come una parte inscindibile dell'ambiente di sviluppo Visual Basic. Questo capitolo spiega come creare add-in con Visual Basic.

Che cos'è un add-in?

Gli add-in sono un modo importante ed entusiasmante di estendere le capacità dell'IDE di Visual Basic. Per esempio, si potrebbero costruire degli add-in per impostare l'aspetto di tutti i form di un progetto o per analizzare e ottimizzare il codice di un progetto. Si possono usare tutte le funzionalità di Visual Basic per interagire con il progetto caricato nell'istanza corrente dell'ambiente di sviluppo VB.

L'ambiente di sviluppo integrato Visual Basic (Visual Basic Integrateci Development Environment) è abbreviato in VBIDE. L'espressione VBIDE. VBE si riferisce a un oggetto che contiene l'istanza corrente dell'ambiente di progettazione Visual Basic. Per ulteriori informazioni sui membri esportati dall'oggetto VBIDE, vedere "I membri dell'oggetto radice" più avanti in questo capitolo.

In realtà, è possibile creare un add-in per Visual Basic usando un linguaggio o un ambiente di sviluppo diversi da VB, per esempio, in Visual C++ o in Delphi32. L'unico requisito è che il linguaggio deve essere capace di creare componenti ActiveX (server OLE) che possano comunicare con istanze dell'oggetto IDE di Visual Basic (VBIDE.VBE). Per una spiegazione dettagliata dei componenti server ActiveX, vedere il Capitolo 23.

Il fatto che si possano creare degli add-in usando Visual Basic stesso, e che non serva nessuna conoscenza di altri linguaggi, apre le porte a tutte le possibilità immaginabili I programmatori VB possono creare add-in per personalizzare i loro stessi ambienti di lavoro e per automatizzare operazioni ripetitive nello sviluppo. Inoltre, non si dovrebbe trascurare la possibilità di distribuire e rivendere ogni add-in che si è scritto.

Con la versione 6 di Visual Basic, il processo di creazione di add-in è cambiato notevolmente. Per chi sapesse già come creare add-in con la versione precedente questo cambiamento può comportare un certo tempo di apprendimento. Ma la buona notizia è che le nuove tecniche sono considerevolmente più facili da usare. Alcuni strati di astrazione nascondono la maggior pane dei dettagli intricati. Inoltre, questa implementazione usa le impostazioni del Registro di configurazione, non gli arcaici file Ini. Perfino per gli sviluppatori, un add-in a VB6 sembrerà una parte integrante di VB.

Gli add-in, indipendentemente dal linguaggio in cui sono stati scritti, sono programmi server ActiveX specializzati, che sono in comunicazione bidirezionale con l'IDE di Visual Basic. Possono venire usati per:

- Automatizzare operazioni ripetitive
- Standardizzare le proprietà di tutti gli oggetti di un progetto o di un form
- Costruire form e applicazioni da un'ossatura standardizzata
- Guidare l'utente in un modo simile ai wizard attraverso quasi qualunque operazione che possa venire compiuta nell'IDE
- Molto altro. Non c'è limite, basta usare la propria immaginazione!

Gli add-in, siccome rilevano gli eventi fatti scattare da istanze di VBIDE.VBE, possono reagire in modo intelligente a ciò che l'utente sta facendo. Per esempio, un add-in può facilmente intervenire quando l'utente fa clic su una voce di menu o apre un form.

Tipi di add-in

Ci sono almeno cinque tipi di add-in per Visual Basic 6:

- L'add-in semplice, indicato semplicemente come add-in. Gli add-in eseguono delle operazioni internamente all'IDE di Visual Basic, spesso in risposta a eventi fatti scattare dalle azioni dell'utente. Queste operazioni possono essere visibili o meno all'utente.
- Il wizard o procedura guidata, che è un add-in implementato con un'interfaccia utente di "wizard". I wizard vengono solitamente usati per guidare gli utenti attraverso compiti complessi ma lineari che possono venire suddivisi utilmente in una sequenza di passi.

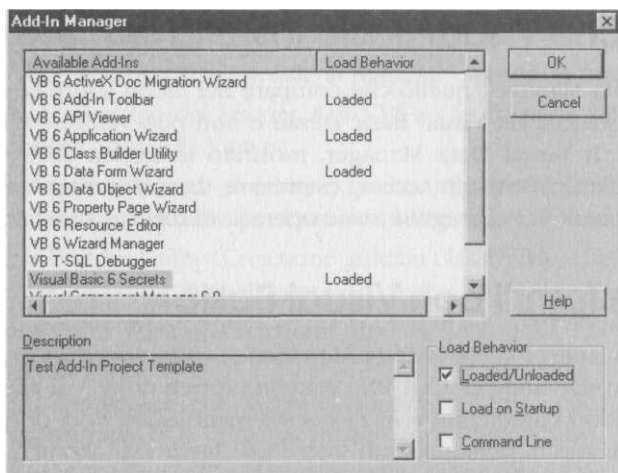
  Per aiutare a creare add-in, VB6 fornisce il Wizard Manager, che è a sua volta un add-in. Il Capitolo 30, spiega in dettaglio come creare un add-in wizard. L'utilizzo di VB per creare un wizard da usarsi all'esterno del VBIDE è stato trattato nel Capitolo 8.

- Una *utility* è un programma in forma di componente ActiveX eseguibile, che può venire eseguito sia come add-in per VB, sia esternamente all'ambiente VBIDE.
- Un *builder* (costruttore) è un tipo di add-in che viene usato per impostare le proprietà di un controllo, o le proprietà che un gruppo di controlli ha in comune. In VB6, la funzionalità di un add-in builder è stata in gran parte rimpiazzata dalle pagine di proprietà personalizzate.
- Un *designer* (progettista) viene usato dagli sviluppatori per creare moduli specializzati.

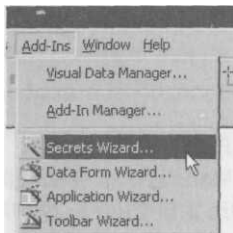
Normalmente, per lanciare un add-in lo si seleziona da un menu o da una toolbar. Tuttavia, non è necessario che un add-in per VB6 venga avviato in questo modo. Un add-in per VB6 può rimanere nascosto sullo sfondo in attesa dello scatto di un evento, come il ridimensionamento di un oggetto. Un add-in non deve necessariamente essere visibile sullo schermo, e non è necessario che l'esecuzione di un add-in produca risultati visibili. Per esempio, un add-in potrebbe azzerare un controllo Timer ogni volta che si carica un nuovo progetto.

Utilizzo dell'Add-In Manager

L'Add-In Manager è una utility che fa parte dell'IDE di Visual Basic e permette di abilitare gli add-in disponibili. Per avviare l'Add-In Manager, lo si sceglie dal menu *Add-Ins.* La finestra di dialogo che comparirà, mostrata in Figura 29.1, permette di caricare (cioè rendere disponibile) o scaricare (cioè disabilitare) gli add-in. Per caricare un add-in, dapprima lo si seleziona nell'Add-In Manager. Successivamente, si imposta la casella *Loaded/Unloaded* nel riquadro *Load Behavior.* Dopo che un add-in è stato caricato, può venire scaricato modificando nuovamente l'impostazione di questa casella di controllo.

Figura 29.1 *L'Add-In Manager è il posto in cui si caricano o si scaricano gli add-in disponibili.*

Si possono usare le impostazioni del riquadro Load Behavior *nell'Add-In Manager per fare in modo che un add-in venga caricato quando si avvia VB, e per renderlo eseguìbile dalla riga di comando.*

Per gli add-in elencati nellAdd-In Manager, sia quelli abilitati che gli altri, ci sono apposite voci nel Registro di configurazione sotto la diramazione HKEY_CURRENT_-USER\Software\Microsoft\Visual Basic\6.0\Addins. Come gli altri server ActiveX la registrazione è per Prog ID. Per un add-in, questo sarà *nome delprogetto.modulo di connessione.*
Gli add-in che sono stati registrati con successo in Windows come componenti ActiveX e sono stati abifitati nell'Add-In Manager solitamente appaiono come voci di menu sul menu *Add-Ins.* Tipicamente, l'utente avvia un add-in facendo clic sulla relativa voce di menu visualizzata nel menu *Add-Ins,* come mostrato nella Figura 29.2. A seconda della progettazione dell'add-in, a questo punto si può attivare un sistema di sottomenu, o caricare un form, oppure eseguire qualche altra azione.

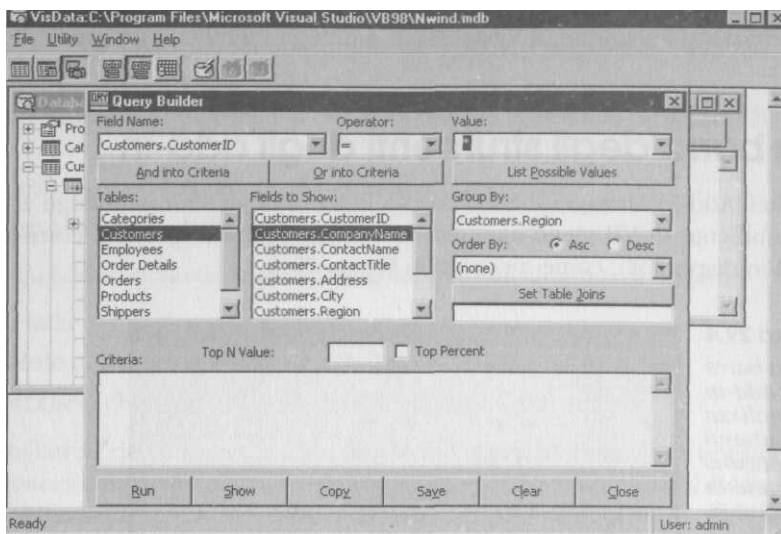
Figura 29.2
Qui il menu Add-Ins comprende numerosiadd-in.

Gli add-in attivi non devono necessariamente venire avviati dal menu Add-Ins. *Si possono usare le proprietà dell'oggetto VBIDE per porre un add-in in qualunque menu di Visual Basic.*

Un add-in che Visual Basic installa sempre

L'add-in Visual Data Manager, quello che compare nel menu sopra la voce *Add-in Manager,* è stato posto lì da Visual Basic stesso e non può venire configurato con l'Add-In Manager. Il Visual Data Manager, mostrato in Figura 29.3, permette di creare nuovi database Microsoft Access, esaminare database in vari altri formati, creare interrogazioni SQL, ed eseguire varie operazioni relative ai database.

Altri add-in forniti con Visual Basic

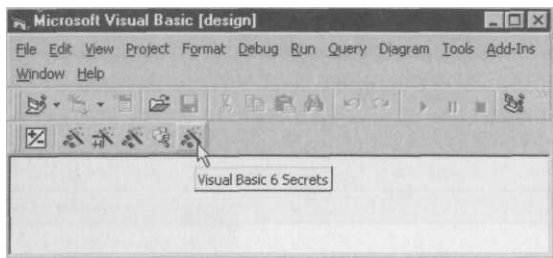
Nel menu *Add-Ins,* sotto la voce *Add-in Manager,* ci sono voci di menu per add-in opzionali che sono stati abilitati. Con VB6 vengono forniti numerosi add-in a cui, se abilitati, corrispondono altrettante voci di questo menu. Altre voci di questo menu corrispondono agli add-in personalizzati, come quelli che ognuno scrive per sé od ottiene da altri sviluppatori di software. Quasi tutti avranno già una certa familiarità con molti degli add-in forniti con VB. Tra di essi vi sono:

Figura 29.3
L'add-in Visual Data Manager serve a creare e manipolare database.

- Package and Deployment Wizard, (Creazione guidata pacchetti di installazione), che crea pacchetti di setup e aiuta a distribuirli. Vedere il Capitolo 28 e il Capitolo 35
- Source Code Control (Controllo del codice sorgente), che aggiunge a Visual Source Safe alcuni progetti scritti usando l'edizione Enterprise di VB6. Vedere il Capitolo 12
- ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX), che aiuta a progettare l'interfaccia dei controlli ActiveX. Vedere il Capitolo 25
- ActiveX Document Migration Wizard (Conversione guidata documenti ActiveX), che aiuta a convenire i form di un progetto in documenti ActiveX. Vedere il Capitolo 28
- Add-In Toolbar (Barra degli strumenti Aggiunte), vedere la sezione successiva
- VB6 API Viewer (Visualizzatore API VB6), che fornisce un'interfaccia fra VB e l'applicazione API Viewer
- VB6 Application Wizard (Creazione guidata applicazioni VB6), che aiuta a costruire l'ossatura iniziale per i progetti VB eseguibili standard
- VB6 Class Builder Utility (Creazione guidata classi VB6), che aiuta a progettare membri dei moduli di classe
- VB6 Data Form Wizard (Creazione guidata form dati), che aiuta a creare form con oggetti associati a una sorgente di dati locale o remota
- VB T-SQL Debugger, che viene usato per aiutare a collaudare e fare il debug di istruzioni SQL immagazzinate (stored procedure)
- VB 6 Wizard Manager (Creazione operazioni guidate VB6), è spiegato nel Capitolo 30

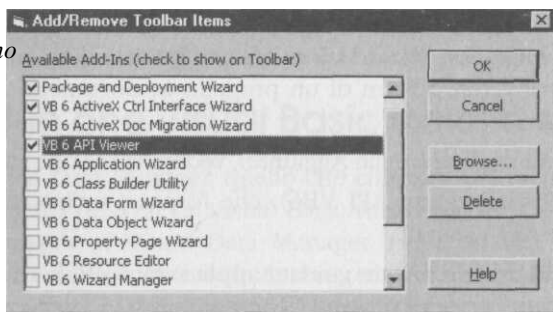
Molti di questi add-in sono estremamente importanti per conto loro, e appaiono essere parte integrante di Visual Basic a tutti gli utenti eccetto quelli di livello avanzato.

La barra degli strumenti degli add-in

Se nell'Add-In Manager viene abilitata la barra degli strumenti degli add-in di VB6 all'ambiente di VB viene aggiunta una barra dotata di pulsanti corrispondenti agli add-in disponibili, come mostrato in Figura 29.4.

Figura 29.4 *La barra degli add-in visualizza dei pulsanti che attivano gli add-in disponibili.*

Come spiegato più avanti in questo capitolo, si può scrivere del codice nei propri add-in personalizzati che inserisca automaticamente un pulsante nella barra degli strumenti degli add-in.

Il pulsante più a sinistra della barra degli add-in apre la finestra di dialogo *Add/ Remove Toolbar Items,* mostrata nella Figura 29.5, che permette agli utenti di configurare la barra degli strumenti.

Figura 29.5 *Gli utenti possono facilmente configurare la barra degli add-in inserendo o togliendo singoli add-in nella finestra di dialogo Add/Remove Toolbar Items.*

Concetti sull'oggetto VBIDE

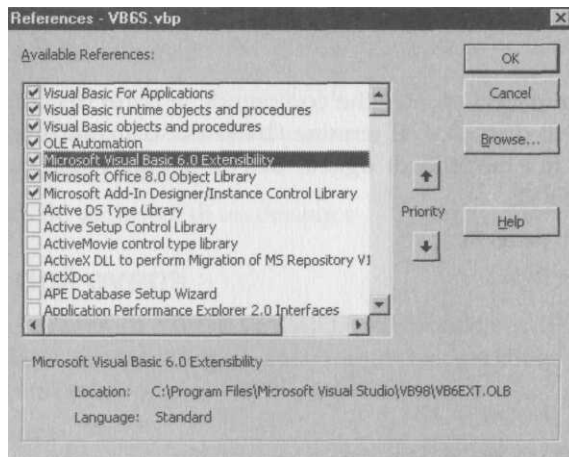
L'oggetto VBIDE (Visual Basic Integrated Development Environment, cioè ambiente integrato di sviluppo Visual Basic) incapsula istanze dell'ambiente Visual Basic esportando i suoi oggetti figli e le loro proprietà, eventi e metodi. Questa esportazione di oggetti, proprietà, eventi e metodi permette di fare molte cose:

- Inizializzare e arrestare gli add-in.
- Aggiungere nuove voci di menu all'ambiente VBIDE, e rispondere quando l'utente seleziona tali voci di menu.
- Gestire la creazione, l'apertura e la chiusura dei file associati al progetto VB corrente.
- Manipolare i form e i loro controlli nel progetto corrente.
- Rispondere alle *azioni* dell'utente.

Lavorare con gli add-in richiede la comprensione di tre distinti tipi di oggetti:

- L'oggetto radice
- Una variabile di istanza di Visual Basic
- AddInInstance, che è un membro di AddInDesignerObjects.

Si possono abilitare i riferimenti a questi oggetti assicurandosi che nella finestra di dialogo *References* del progetto siano selezionate le voci *Microsoft Visual Basic 6.0 Extensibility, Microsoft Office 8.0 Object Library,* e *Microsoft Add-in Designer/ Instance Control Library,* come mostrato nella Figura 29.6.

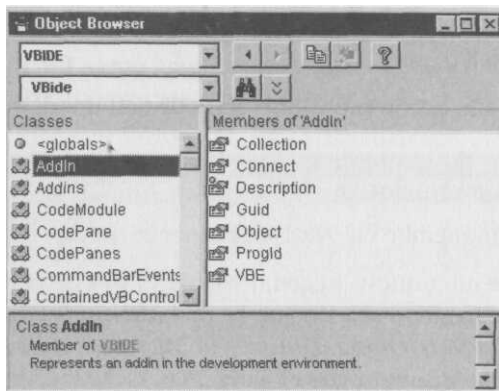
Figura 29.6 *Per lavorare con gli oggetti necessari per creare un add-in, i riferimenti a tali oggetti devono venire abilitati usando la finestra di dialogo References del progetto.*

L'oggetto radice (root) rappresenta un'istanza dell'ambiente Visual Basic, e comprende tutti i relativi file sorgente di programma. Per farvi riferimento, si usa una variabile di istanza di Visual Basic. Per esempio, la seguente dichiarazione assegna l'istanza corrente dell'ambiente VB alla variabile globale *gMyVBInstance:*

```
Global gMyVBInstance As VBIDE.VBE
```

I membri dell'oggetto radice

Per far riferimento ai membri dell'oggetto radice, si usano l'operatore punto e una variabile di istanza di Visual Basic.

Si può usare l'Object Browserper visualizzare i membri dell'oggetto VBIDE, come mostrato in Figura 29.7. Perfare ciò, ci si assicuri che nellafinestra di dialogo References *sia selezionata la voce* Microsoft Visual Basic 6.0 Extensibility. *Poi si apra l'Object Browser, e si selezioni VBIDE nella casella di riepilogo a discesapiù in alto*

Figura 29.7
Si può usare l'Object Browser per visualizzare i membri dell'oggetto VBIDE, purché sia stata abilitata la libreria Microsoft Visual Basic 6.0 Extensibility.

Gli oggetti del modello di estendibilità di VBIDE.VBE si raggnippano nelle seguenti categorie:

- Estensione dell'interfaccia utente, che comprende le barre dei comandi, le finestre e i pannelli di codice. Il termine "barra dei comandi" *(command bar)* viene usato nel modello di oggetti dell'estendibilità per indicare le toolbar e i menu di VB.
- Manipolazione dei progetti
- Manipolazione dei form
- Risposta agli eventi
- Manipolazione del codice
- Add-in dell'utente

Estensione dell'interfaccia utente

Gli oggetti che estendono l'interfaccia utente sono le collezioni (insiemi) `CommandBars`, `Windows`, e `CodePanes`. Queste collezioni vengono usati per fare cose come:

- Aggiungere un nuovo pulsante o comando a una toolbar o menu esistente.
- Creare un menu o una toolbar completamente nuovi per un add-in.
- Aprire, chiudere, spostare, o ridimensionare una finestra dell'interfaccia utente.
- Usare la collezione `CodePanes` e l'oggetto `CodePane` per visualizzare del codice e per determinare quale codice è stato selezionato dall'utente.

Manipolazione dei progetti

L'insieme VBProjects consente di manipolare progetti VB in molti modi, tra cui:

- Selezionare da un gruppo un progetto su cui operare, o selezionarne alcuni per applicare a tutti le stesse operazioni
- Togliere tutti i progetti dall'ambiente Visual Basic e iniziare un nuovo progetto
- Aggiungere nuovi progetti all'attuale sessione Visual Basic
- Reagire appropriatamente al caricamento o allo scaricamento di un progetto particolare, per esempio notificando tali eventi ai propri add-in
- Visualizzare i nomi dei progetti, per esempio, in una casella combinata *(combo box)*
- Modificare le opzioni di un progetto

Manipolazione dei form

Gli oggetti che appartengono all'oggetto VBForm permettono di manipolare da programma i designer di un form, di un controllo, di una pagina di proprietà, o di un documento ActiveX per eseguire varie operazioni, tra cui:

- Aggiungere un designer
- Aggiungere del codice o dei controlli
- Nascondere o visualizzare un designer
- Posizionare dei controlli
- Modificare le proprietà di un designer

Risposta agli eventi

La collezione `Events` di VBIDE.VBE aiuta a rispondere agli eventi che avvengono nell'istanza di Visual Basic. Si può anche avere bisogno di gestire eventi che avvengono in altri insiemi di oggetti VBIDE, tra cui:

- `VBProjects`
- `VBComponents`
- `VBControls`

Manipolazione del codice

I membri dell'oggetto `CodeModule` di VBE mettono nella condizione di controllare e manipolare da programma il codice in Visual Basic. Usando questi oggetti, si possono eseguire molte operazioni, tra cui:

- Selezionare, aggiungere o eliminare righe di codice
- Cercare e/o sostituire Decorrenze di stringhe specifiche

L'oggetto AddInInstance

L'oggetto AddInInstance, membro di AddInDesignerObjects, consente di rilevar e inserire codice negli eventi di connessione e sconnessione degli add-in:

- OnConnection è il metodo che inserisce l'add-in nell'IDE di Visual Basic.
- OnDisconnection (alla sconnessione) toglie l'add-in da Visual Basic.

Nella prossima sezione verrà spiegato dettagliatamente il procedimento di creazione di un add-in.

Creazione di un semplice add-in

Per fare i primi passi nel procedimento di creazione di un add-in per VB, si può usare un designer speciale, AddIn Class (classe di add-in), per fare prima. Per aggiungere al proprio progetto un'istanza di questo designer, lo si deve abilitare nella scheda *Designers* della finestra di dialogo *Components,* come mostrato nella Figura 29.8.

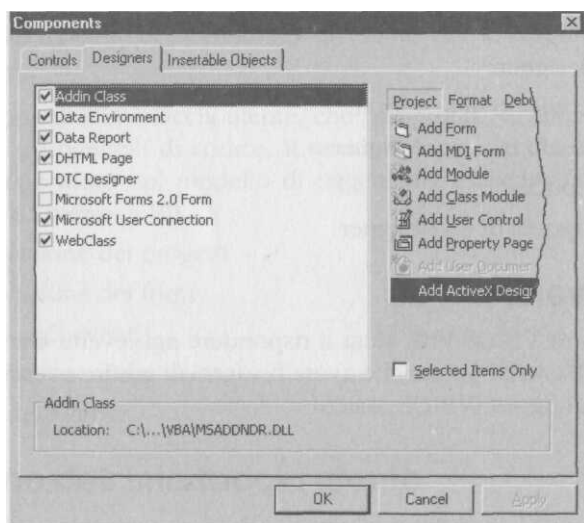
Figura 29.8
Si può usare la scheda Designers della finestra di dialogo Components per abilitare il designer AddIn Class.

Successivamente, si devono inserire alcune informazioni nella finestra del designer, mostrata in Figura 29.9. A questo punto, il designer comparirà nel Project Explorer, come mostrato in Figura 29.10. Se si visualizza il codice che è stato posto nel modulo del designer, si troverà lo stretto necessario per un add-in.

Figura 29.9

Usando il designer AddIn Class, si possono inserire informazioni per il proprio add-in.

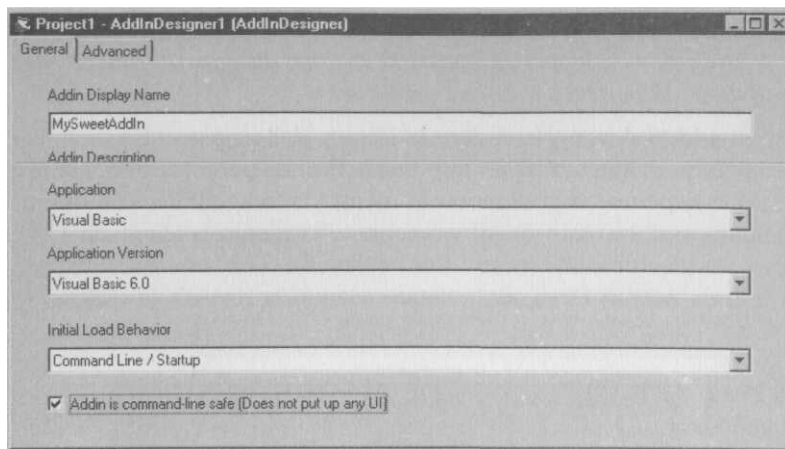



Figura 29.10

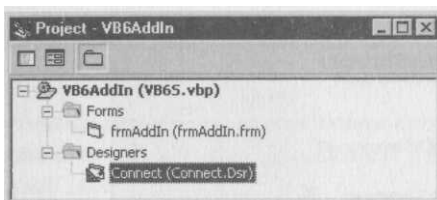
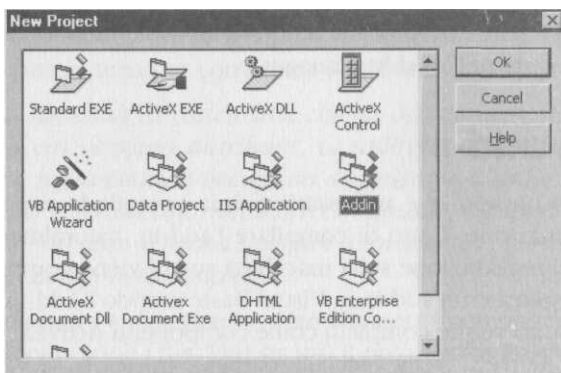
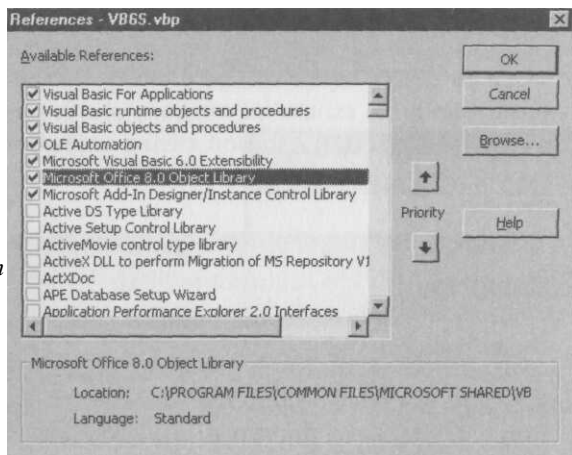
Per visualizzare il codice modello generato dal designer AddIn Class, si usi l'opzione View Code del Project Explorer.

Un altro modo per giungere più o meno allo stesso risultato è creare un nuovo progetto di add-in selezionando AddIn *dalla finestra di dialogo* New Project, *come mostrato nella Figura 29.11.*

Figura 29.11

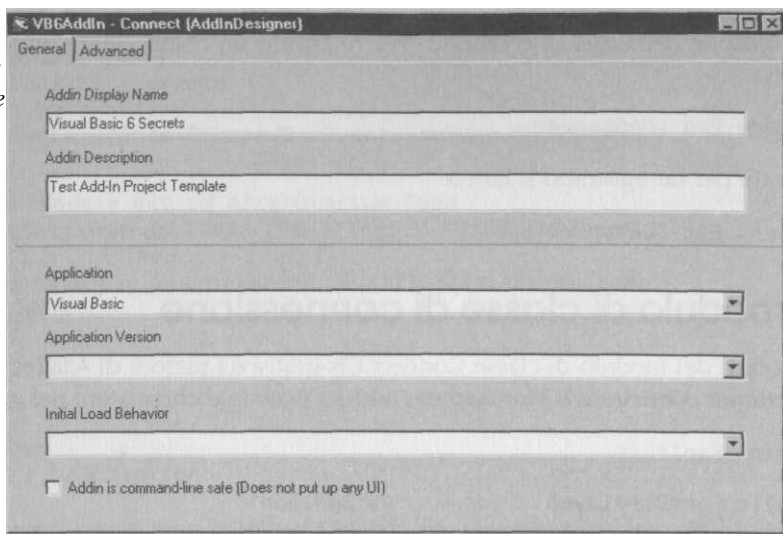
Se si seleziona Addin dalla finestra di dialogo NewProject, verrà creato un progetto contenente il codice modello necessario per un add-in.

È più facile capire il procedimento di creazione di un add-in se si guarda un esem= pio. Il progetto d'esempio è salvato sul CD-ROM allegato al libro col nome VB6s.Vbp Contiene un designer di add-in e un form.

Tutti gli add-in devono includere la libreria dell'oggetto Microsoft Visual Basic 6.0 Extensibility. Inoltre, chi vorrà fare in modo che i propri add-in, come è tipico, inse= riscano le rispettive voci di menu in menu VB, o aggiungano pulsanti a barre degli strumenti, avrà bisogno di includere un riferimento a Microsoft Office 8.0 Object Library. La Figura 29.12 mostra dei riferimenti a entrambe le librerie, come anche alla libreria Add-in Designer, abilitate usando la finestra di dialogo *References* del progetto.

Figura 29.12 *Si devono aggiungere gli oggetti Microsoft Visual Basic 6.0 Extensibility e la Microsoft office 8,0 Object Library al proprio progetto di add-in usando la finestra di dialogo References.*

Se si fa doppio clic sul designer di add-in nel Project Explorer, la finestra di dialogo mostrata in Figura 29.13 permetterà di introdurre il nome e la descrizione che gli utenti vedranno visualizzata nell'Add-In Manager.

Compilare un add-in

Prima di poter provare un add-in è necessario prima compilarlo e poi registrarlo sulla macchina di destinazione. L'atto di compilare l'add-in, naturalmente, gestisce automaticamente la sua registrazione sulla macchina su cui viene compilata.
Adesso proveremo a registrare un'add-in in Visual Basic usando l'Add-In Manager.
I progetti di add-in devono venire compilati come componenti ActiveX, nel senso di DLL ActiveX o di EXE ActiveX. Nella "vecchia parlata", questa frase va riformulata così: gli add-in vengono compilati come server OLE in-process (.Dll) o out-of-process (.Exe). Per determinare in quale modo verrà compilato un progetto, si usa la casella di riepilogo a discesa *Project Type* sulla scheda *General* della finestra di dialogo *Project Properties.* La cosa più comune è compilare gli add-in come DLL ActiveX. Come regola generale, i componenti in-process, cioè le DLL ActiveX, sono più veloci dei componenti out-of-process (cioè gli EXE ActiveX).

Figura 29.13 *Il designer di add-in permette di introdurre il nome e la descrizione dell'add-in come apparirà nell'Add-In Manager.*

La ragione primaria per cui si potrebbe volere compilare un add-in come EXE ActiveX è di permettere all-add-in di funzionare indipendentemente da un'istanza dell'IDE di Visual Basic.

Registrazione e deregistrazione manuale degli add-in

Quando si compila un'applicazione di componente ActiveX, automaticamente si registra l'applicazione nel sistema su cui viene compilata. Tuttavia, purtroppo il debug e il collaudo di un programma comportano il tornare numerose volte a modificare il programma per correggere i problemi.

Vi è anche la necessità di assicurarsi che gli add-in vengano registrati sui sistemi degli utenti a cui vengono distribuiti. Le utility di setup, tra cui c'è l'Application Setup Wizard, solitamente si occupano di generare il codice necessario. Si veda il Capitolo 35, per ulteriori informazioni. Comunque, è importante sapere come farlo manualmente, nel caso si debba spiegare a un utente come registrare o deregistrare uno dei proprì componenti ActiveX.

Ecco come registrare e deregistrare manualmente i componenti ActiveX. Per i componenti in-process (cioè i file .Dll) sia usa il programma Regsvr32.Exe. Per esempio, per registrare un server in-process si lancia:

```
Regsvr32  MyServ.Dll
```

Mentre per deregistrare lo stesso server si lancia:

```
Regsvr32 /u MyServ.Dll
```

Per registrare e deregistrare i componenti out-of-process (cioè i file .Exe) si usa un'opzione della riga di cornando. Per registrare un componente out-of-process si lancia:

```
MyServ.Exe /regserver
```

Mentre per deregistrarlo si lancia:

```
MyServ.Exe /unregserver
```

Il modulo di classe di connessione

Il codice del modulo di classe Connect.Cls gestisce i metodi di AddInlnstance utilizzati per connettere e sconnettere l'add-in. Ecco le dichiarazioni del modulo:

```
Option Explicit

Public FormDisplayed            As Boolean
Public VBInstance               As VBIDE.VBE
Dim mcbMenuCommandBar           As Office.CommandBarControl
Dim mfrmAddIn                   As New frmAddIn
Public WithEvents MenuHandler   As CommandBarEvents
```

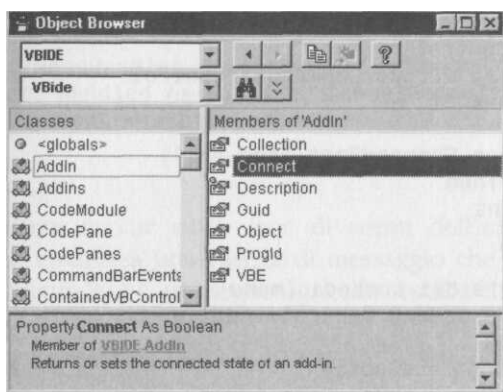
La routine OnConnection aggiunge l'add-in a VB, come mostrato nel Listato 29.1.

Listato 29. i *La routine OnConnection di IDTExtensibility.*

```
'questo metodo aggiunge l'Add In a VB

Private Sub AddinInstance_OnConnection(ByVal _
   Application As Object, ByVal ConnectMode As _
       AddInDesignerObjects.ext_ConnectMode,_
       ByVal AddInInst As Object, custom() As Variant)
   On Error GoTo error_handler
   MsgBox "I've been connected!"
   'salva l'istanza di VB
   Set VBInstance = Application

   'questo è un buon punto per impostare una interruzione e
   'valutare vari oggetti, proprietà e metodi dell'add-in
   Debug.Print VBInstance.FullName

   If ConnectMode = ext_cm_External Then
      'Usato dalla barra degli strumenti wizard per avviare
      'questo wizard
      Me.Show
```

```
    Else
       Set mcbMenuCommandBar = AddToAddInCommandBar _
          ("VBSSecrets")
       'sincronizza l'evento
       Set Me.MenuHandler =
       VBInstance.Events.com/nandBarEvents (mcbMenuCommandBar)
    End If

    If ConnectMode = ext_cm_AfterStartup Then
       If GetSetting(App.Title, "Settings", "DisplayOnConnect", _
          "0") = "1" Then
          'imposta per visualizzare il form alla connessione
          Me.Show
       End If
    End If

    Exit Sub

error_handler:
    MsgBox Err.Description
End Sub
```

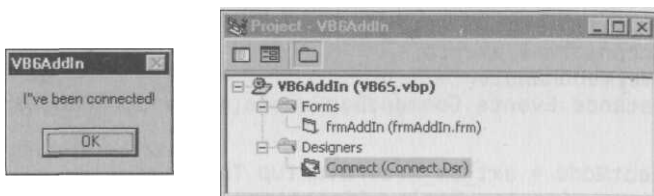
La costante ConnectMode viene passata come parametro alla routine OnConnection. Come mostrato in Figura 29.14, si può usare l'Object Browser per determinare quali sono i possibili valori dei membri di ConnectMode, e ciò che significano tali valori.

Figura 29.14 *Se nell'Object Browser si seleziona l'oggetto VBIDE, si viene aiutati a comprendere i significati delle costanti che rappresentano i possibili valori delle classi, delle collezioni, e dei membri di tale oggetto.*

Dopo che l'add-in è stato abilitato nell'Add-In Manager, la routine OnConnection scatta quando l'utente fa clic su *OK* nell'Add-In Manager.

Purché l'add-in sia abilitato, la sua routine OnConnection *scatta anche quando viene avviata un'istanza di Visual Basic stesso.*

La Figura 29.15 mostra la casella di messaggio visualizzata dalla routine OnConnection per dimostrare che è stata lanciata.

Figura29.15

Se si aggiunge un'istruzione MsgBox alla routine OnConnection, si può avere un'idea di quando viene invocata questa routine.

All'interno della routine OnConnection, viene aggiunta al menu Add-Ins di Visual Basic una voce relativa al nuovo add-in. Questo si ottiene invocando la funzione AddToAddInCommandBar:

```
Set mcbMenuCommandBar = AddToAddInCommandBar
    ("VB6 Secreta AddIn")
```

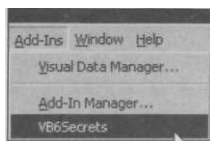
Il Listato 29.2 contiene la funzione AddToAddInCommandBar, pure del modulo di classe Connect, che si può personalizzare per adattarla alle necessità dei propri add-in:

Listato 29.2 *Aggiunta di una voce di menu per un add-in.*

```
Function AddToAddInCommandBar(sCaption As String) _
    As Office.CommandBarControl
    Dim cbMenuCommandBar As Office.CommandBarControl
    'oggetto barra dei comandi
    Dim cbMenu As Object
    Set cbMenu = VBInstance.CommandBars("Add-Ins")
    If cbMenu Is Nothing Then
        'nessun menu Add-Ins
        Exit Function
    End If
    'lo aggiunge alla barra dei comandi (menu)
    Set cbMenuCommandBar = cbMenu.Controls.Add(1)
    'imposta la didascalia
    cbMenuCommandBar.Caption = sCaption
    Set AddToAddInCommandBar = cbMenuCommandBar
End Function
```

L'oggetto CommandBar *contiene altri oggetti CommandBar che possono agire come pulsanti o come comandi di menu. In altre parole, se l'oggetto* CommandBar *avesse un nome che indica ciò che contiene, verrebbe chiamato* CommandBarAndMenuObjectHolder *(contenitore di oggetti barra dei comandi e menu).*

Come mostrato in Figura 29.16, quando la routine OnConnection chiama la funzione AddToAddInCommandBar, viene invocato il metodo Add dell'oggetto CommandBars per aggiungere al menu *Add-Ins* una voce avente come didascalia il parametro passato alla funzione.

Figura 29.16
Si può usare CommandBars per creare una voce di menu per il proprio add-in.

Una routine di nome MenuHandler è stata dichiarata con la clausola WithEvents nella sezione Declarations del modulo di classe:

```
Public WithEvents MenuHandler As CommandBarEvents
```

Il nuovo oggetto barra dei comandi (o voce di menu) è stato assegnato alla variabile mcbMenuCommandBar:

```
Set mcbMenuCommandBar = AddToAddInCommandBar
    ("VB6 Secrets AddIn")
```

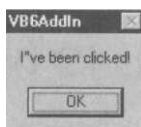
La seguente riga di codice della routine OnConnection dice a VB dove mandare gli eventi generati dalla nuova voce di menu:

```
Set Me.MenuHandler =
   VBInst.Events.CommandBarEvents(mcbMenuCommandBar)
```

Si dovrà aggiungere una routine corrispondente al modulo di classe Connect per gestire gli eventi che scattano quando gli utenti fanno clic sulla nuova voce di menu:

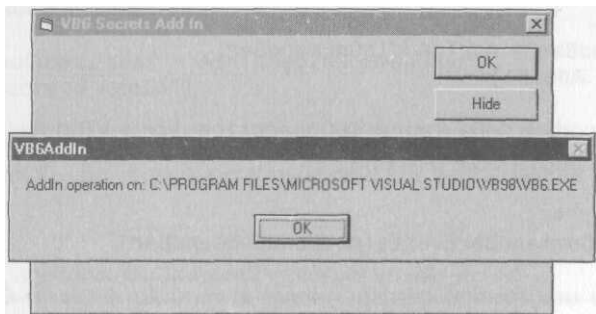
```
Private Sub MenuHandler_Click(ByVal CommandBarControl _
   As Object, handled As Boolean, CancelDefault As Boolean)
   MsgBox "My menu was clicked!"
   Me.Show
End Sub
```

Quando l'utente fa clic sulla voce di menu dell'add-in, la routine Click del MenuHandler visualizza una casella di messaggio che indica che è scattata, come mostrato in Figura 29.17, ed esegue ogni altro codice che è stato posto in essa.

Figura 29.17
Casella di messaggio aggiunta al gestore dell'evento Click.

Il gestore dell'evento Click della voce di menu dell'add-in è il punto in cui normalmente verrebbe posto il codice che realizza le funzioni a cui è destinato l'add-in. Questo è il punto in cui si visualizzerebbe un form, o, nel caso l'add-in non avesse un aspetto visibile, si eseguirebbe un'altra operazione.
L'istruzione alla fine della routine MenuHandler, Me.Show, invoca la routine Show del modulo di classe Connect, che visualizza un'istanza del form che fa parte del progetto di add-in VB6S:

```
Sub Show()
   On Error Resume Next
   If mfrmAddIn Is Nothing Then
      Set mfrmAddIn = New frmAddIn
   End If
   Set mfrmAddIn.VBInstance = VBInstance
   Set mfrmAddIn.Connect = Me
   FormDisplayed = True
   mfrmAddIn.Show
End Sub
```

Verrà visualizzata un'istanza del frmAddIn. Come mostrato nella Figura 29.18, ha un pulsante *OK* e un pulsante *Hide* (nascondi).

Figura 29.18
Solitamente, si vorrà visualizzare un form quando l'utente fa clic sulla voce di menu del proprio add-in.

Facendo clic sul pulsante OK, si visualizza il valore della proprietà FullName dell'istanza corrente di VB:

```
Private Sub OKButton_Click()
   MsgBox "Attività dell'Add-In su: " & VBInstance.FullName
End Sub
```

Facendo clic sul pulsante *Hide,* si invoca la routine Hide del modulo di classe Connect, che chiama il metodo Hide dell'istanza di form:

```
Private Sub cmdHide_Click()
   Connect.Hide
End Sub

Sub Hide()
   On Error Resume Next
   FormDisplayed = False
   mfrmAddIn.Hide
End Sub
```

La routine OnStartupComplete può venire usata per visualizzare un'istanza del form dell'add-in quando l'IDE di Visual Basic finisce di caricarsi. La routine OnAddInsUpDate scatta quando vengono salvate le modifiche apportate con l'Add-In Manager. Il Listato 29.3 mostra la routine OnDisconnection, che scatta quando l'add-in viene tolto da VB. Questo punto è quello in cui si dovrebbe porre il codice di pulizia finale.

Listato 29.3 *Sconnessione di un add-in.*

```
'questo modo elimina l'Add-In da VB.

Private Sub AddinInstance_OnDisconnection(ByVal RemoveMode _
    As AddInDesignerObjects.ext_DisconnectMode, _
    custom() As Variant)
    On Error Resume Next

    MsgBox "I've been disconnected! "

    ' elimina la voce della barra dei comandi
    mcbMenuCommandBar.Delete

    'chiude l'Add-In
    If FormDisplayed Then
        SaveSetting App.Title, "Settings", "DisplayOnConnect", "1"
        FormDisplayed = False
    Else
        SaveSetting App.Title, "Settings", "DisplayOnConnect", "0"
    End If

    Unload mfrmAddIn
    Set mfrmAddIn = Nothing
End Sub
```

La casella di messaggio inclusa nella routine OnDisconnection (Figura 29.19) viene mostrata quando l'utente disabilita l'add-in e fa clic su *OK* nell'Add-In Manager, o quando un'istanza di VB viene portata a termine con tutti gli add-in ad essa connessi.

Figura 29.19 *La routine OnDisconnection scatta quando l'add-in viene disconnesso.*

L'add-in Change Colors

Il resto di questo capitolo dimostra come creare un add-in pienamente funzionale chiamato Change Colors (cambia colori). Questo add-in permette agli utenti di cambiare la proprietà BackColor o la proprietà ForeColor di tutti i controlli che hanno tali proprietà, facendo un solo clic. L'add-in modificherà le proprietà di tutti i controlli presenti su tutti i componenti aperti in cui possono risiedere controlli, in tutti i progetti che sono caricati nell'istanza corrente di Visual Basic.

Questo add-in può risultare molto comodo quando si progettano form complessi con molti controlli, per i quali si vuole avere lo stesso schema di colori. Eseguendo l'add-in Change Colors, si possono facilmente modificare i colori di tutti i controlli di un progetto.

Per modificare le proprietà dei controlli, il programma deve navigare nella gerarchia di oggetti di VBIDE.VBE. Ciò significa scendere dall'istanza corrente di VB a tutti i progetti che sono caricati, da qui a tutti i componenti aperti con designer, e così via, come vedremo in dettaglio. La strategia di navigazione *applicata* nel codice può essere considerata un'opportunità per comprendere meglio questa importante struttura.

L'add-in funziona bene così com'è scritto, ma si potrebbero anche introdurre abbellimenti e raffinamenti. Con gli add-in e con l'oggetto VBIDE si possono fare molte altre cose, oltre a quelle dimostrate in questo capitolo.

Change Colors funziona così: quando lo si sceglie dal menu *Formai* di VB, compare una finestra di dialogo che permette di scegliere le proprietà BackColor o ForeColor da applicare ai controlli del form. Si usa il controllo Common Dialog, nella sua modalità ShowColor (mostra colore), per selezionare effettivamente i colori.

Sipossono mettere voci di menuper ipropri add-in in qualsivogliapunto della struttura dì menu di VB. Il menu Format *è sembrato appropriato per Change Colors perché le altre voci di* Format *hanno a che fare con l'aspetto dei controlli e dei componenti.*

Il progetto dell'add-in Change Colors è salvato sul CD-ROM allegato al libro col nome Colors. Vbp. È stato compilato in codice nativo come DLL ActiveX. Affinchè l'add-in compaia nell'Add-In Manager, deve venire registrato sul proprio sistema.

Dopo aver compilato il progetto, si può caricare l'add-in Change Colors nell'Add-In Manager, come mostrato in Figura 29.20. Dopo che è stato caricato, lo si può selezionare dal menu *Format* di Visual Basic, come mostrato nella Figura 29.21.

Figura 29.20
Si può caricare l'add-in Change Colors dall'Add-In Manager.

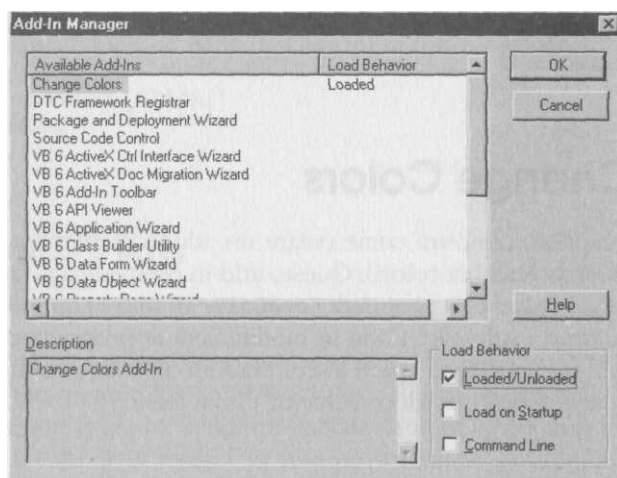
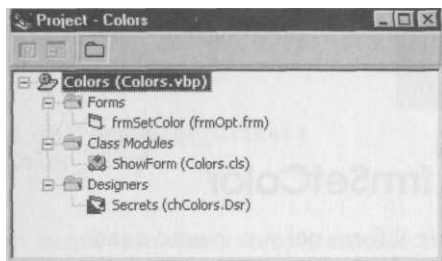



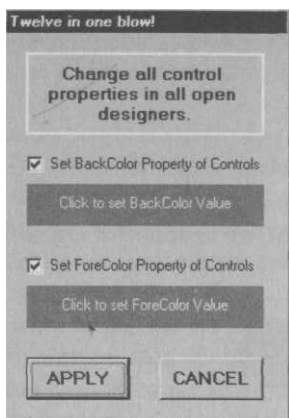
Figura 29.21
L'add-in Cbange Color parte dal menu Add-Ins.

Ci sono tre oggetti nel progetto Colors: un form, un modulo di classe, e il designer di add-in (vedere la Figura 29.22).

Figura 29.22
Il progetto Colors comprende un form, un modulo di classe, e il designer di add-in.

Come far funzionare Change Colors

Questa sezione descrive come l'effettivo add-in Change Colors viene messo insieme, nel senso di come si fa il lavoro di modificare tutti i controlli. Più avanti torneremo alla meccanica dell'inserimento dell'add-in nell'ambiente VB.

Dapprima, creeremo il form, frmSetColor, che useremo per consentire agli utenti di specificare i colori e le proprietà che vogliono modificare: nell'esempio, ForeColor e/o BackColor. La Figura 29.23 visualizza questo form in fase di esecuzione come è stato progettato. C'è un controllo Common Dialog aggiunto al form che non si può vedere in fase di esecuzione, e un'etichetta invisibile di nome RetVal. RetVal rende informazioni di stato quando il form viene istanziato, come spiegato nel Capitolo 3. Quando si fa clic sul pulsante *Apply,* il valore di RetVal diventa quello della costante VBA vbOK; quando si fa clic su *Cancel,* il valore di RetVal diventa vbCancel. Questo form usa dei controlli casella di immagine (PictureBox) per la selezione e visualizzazione dei colori. Avrei preferito usare i pulsanti di comando, ma questi controlli non hanno la proprietà ForeColor e la loro proprietà BackColor non ha effetto visivo. Con un'etichetta sovrapposta a una casella di immagine, si può simulare un pulsante di comando che visualizza le proprietà ForeColor e BackColor.

Figura 29.23 *Il form delle opzioni dell'utente nell'add-in Change Color permette agli utenti di cambiare le proprietà di colore di tutti i controlli contenuti nei designer aperti.*

Programmazione di frmSetColor

Dapprima, si dovrebbe inizializzare il form nel suo evento Load:

```
Private Sub Form_Load()
   Dim fcolor As Long
   Dim bcolor As Long
   With Screen
      Left = (.Width - Width) / 2
      Top = (.Height - Height) / 2
   End With
   chkForeColor.Value = 1    ' Applica alle proprietà ForeColor
   chkBackColor.Value =1     'e BackColors
   fcolor = vbYellow         ' inizializza a giallo
   bcolor = vbRed            ' inizializza a rosso
   SetPicts fcolor, bcolor
End Sub
```

La routine SetPicts (imposta immagini), chiamata sia dall'evento Load del form che quando l'utente modifica la selezione del colore, imposta i valori di colore delle caselle di immagine, che sono pulsanti simulati come appena descritto, a seconda dei parametri passati.

```
Private Sub SetPicts(ByVal fcolor As Long, ByVal bcolor As Long)
   pctBackColor.BackColor = bcolor
   pctForeColor.BackColor = bcolor
   lblBackColor.ForeColor = fcolor
   lblForeColor.ForeColor=fcolor
End Sub
```

Quando l'utente fa scattare l'evento Click di una delle caselle di immagine per modificare le impostazioni dei colori, viene chiamata la routine ChooseColor (scegli colore):

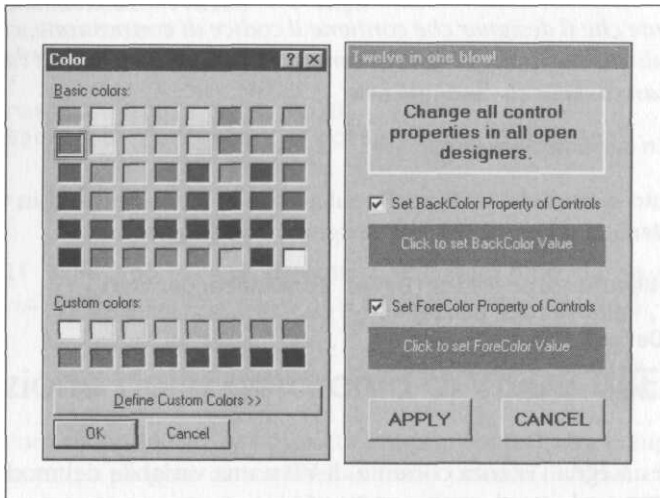
```
Private Sub pctBackColor_Click()
   ChooseColor "B", lblBackColor.ForeColor, _
```

```
      pctBackColor.BackColor
End Sub
Private Sub pctForeColor_Click()
   ChooseColor "F", lblForeColor.ForeColor, _
      pctBackColor.BackColor
End Sub
```

Per completare la simulazione del pulsante, si dovrebbe fare in modo che un clic sulle etichette sovrapposte alle caselle di immagine abbia lo stesso effetto di un clic sulla sottostante casella di immagine. Per ottenere ciò, basta aggiungere una chiamata al gestore della casella di immagine dal gestore dell'etichetta:

```
Private Sub lblBackColor_Click()
   pctBackColor_Click
End Sub

Private Sub lblForeColor_Click()
   pctForeColor_Click
End Sub
```

ChooseColor stessa inizializza semplicemente il controllo Common Dialog con il valore del colore corrente, chiama il dialogo comune usando il metodo ShowColor, e rende il valore che l'utente seleziona, purché non sia stato premuto *Cancel,* come mostrato in Figura 29.24. Le finestre di dialogo comuni sono trattate nel Capitolo 7.

Figura 29.24 *La routine ChooseColor usa il controllo Common Dialog per fare in modo che l'utente selezioni un colore.*

Per determinare a quale casella di immagine appartiene il gestore dell'evento Click che ha chiamato la routine, viene usata una struttura di controllo di salto, descritta nel Capitolo 4.

```
Private Sub ChooseColor(Which As String, ByVal fcolor As Long,
   ByVal bcolor As Long)
   CommonDialog1.CancelError = False
   On Err GoTo ErrHandler
```

```
    If Which = "B" Then
        CommonDialog1.Color = bcolor
    Else
        CommonDialog1.Color = fcolor
    End If
    CommonDialog1.ShowColor
    If Which = "B" Then
        bcolor = CommonDialog1.Color
    Else
        fcolor = CommonDialog1.Color
    End If
    SetPicts fcolor, bcolor
    Exit Sub
ErrHandler:
    Exit Sub
End Sub
```

Questa è tutta la logica del codice incapsulato in frmSetColor; sebbene, come vedremo fra un attimo, ci sia dell'altro. La logica di decisione in frmSetColor è in realtà controllata dal modulo di classe che lo istanzia in conseguenza dello scatto dell'add-in Change Colors.
In altre parole, quando l'utente fa clic sulla voce di menu dell'add-in, viene chiamata una routine di nome AfterClick, che si trova nel modulo di classe ShowForm, salvato col nome Color.Cls. Comunque, prima o poi arriveremo a parlare del funzionamento interno di questo meccanismo.

Si tenga presente che il designer che contiene il codice di connessione, che si chiama Secrets ed è salvato col nome chColors.Dsr sul CD-ROM, dichiara mfrmAddIn *come una nuova istanza della classe* ShowForm:

```
Dim mfrmAddIn As New ShowForm
```

L'evento scattato quando l'utente fa clic sulla voce di menu dell'add-in viene gestito dalla routine MenuHandler_Click nel designer di add-in Secrets:

```
Private Sub MenuHandler_Click(ByVal CommandBarControl _
    As Object, handled As Boolean, _
        CancelDefault As Boolean)
    mfrmAddIn.VBIDE = VBInstance
    mfrmAddIn.AfterClick
End Sub
```

Questo codice assegna l'istanza corrente di VB a una variabile del modulo di classe ShowForm, VBIDE, e chiama la routine AfterClick.
Ecco le dichiarazioni generali per il modulo di classe ShowForm:

```
Option Explicit
Dim X As New frmSetColor
Dim ThisVBInstance As VBide.Application
```

X verrà usato per istanziare frmSetColor per l'introduzione delle preferenze dell'utente. ThisVBInstance verrà usato per immagazzinare il valore della proprietà VBide dopo che è stato passato dal gestore di menu nel modulo di classe AddIn.

In questo caso, non c'è bisogno della routine Property Get. C'è bisogno solamente della routine Property Let VBide, che è ciò che consente al modulo AddIn di immagazzinare il valore dell'istanza di VB nella variabile VBInstance, a livello del modulo ShowForm:

```
public Property Let VBide(vNewValue)
   Set ThisVBInstance = vNewValue
End Property
```

Siccome la routine Property Get VBide è vuota, si può eliminare il suo codice di intelaiatura.
Il codice di AfterClick di ShowForm istanzia frmSetColor, lo popola con valori di default, e chiama SetFormsAndControls (imposta i form e i controlli), se l'utente ha fatto clic sul pulsante *Apply*.

```
Public Sub AfterClick()
   Dim Active, SetF, SetB As Boolean
   X.Show vbModal
   If X.RetVal = vbOK Then
      SetF = False
      SetB = False
      If X.chkForeColor.Value = 1 Then
         'Applica alle proprietà ForeColor
         SetF = True
      End If
      If X.chkBackColor.Value = 1 Then
         'e a BackColors
         SetB = True
      End If
      SetFormsAndControls SetF, SetB, _
      X.lblBackColor.ForeColor, X.pctBackColor.BackColor
   End If
   Unload X
End Sub
```

L'istruzione If X.RetVal = vbOK determina se è stato fatto clic su Apply usando l'etichetta invisibile RetVal. Questa tecnica è spiegata nel Capitolo 3.

Esplorazione della gerarchia di VBIDE.VBE

L'effettivo lavoro di modificare le proprietà di tutti i controlli sui componenti aperti in designer in tutti i progetti caricati nell'istanza corrente di VB è effettuato dalla routine SetFormsAndControls (vedere il Listato 29.4). SetFormsAndControls è un membro privato della classe ShowForm, e viene chiamata quando si fa clic sul pulsante *Apply*.

Listato 29.4 *Modifica delle proprietà di tutti i controlli.*

```
Private Sub SetFormsAndControls(ByVal YesToFore _
    As Boolean, ByVal YesToBack As Boolean, _
    ByVal frcol As Long, ByVal bkcol As Long)
    Dim AllProjects As VBProjects
    Dim ThisVBProject As VBProject
    Dim TheC'jrrentComponent As VBComponent
    Dim ThisControl As VBControl

    On Error Resume Next
    '*** Può darsi che il controllo non abbia le proprietà BackColor
    '*** o ForeColor!' Se non si trovano procede.
    Set AllProjects = ThisVBInstance.VBProjects
    For Each ThisVBProject In AllProjects
        For Each TheCurrentComponent In ThisVBProject.VBComponents
            If TheCurrentComponent.HasOpenDesigner Then
                For Each ThisControl In
                    TheCurrentComponent.Designer.VBControls
                    If YesToBack Then
                        ThisControl.Properties("BackColor") = bkcol
                    End If
                    If YesToFore Then
                        ThisControl.Properties("ForeColor") = frcol
                    End If
                Next ThisControl
            End If
        Next TheCurrentComponent
    Next ThisVBProject
    Set AllProjects = Nothing
    Set ThisVBProject = Nothing
    Set TheCurrentComponent = Nothing
    Set ThisControl = Nothing
End Sub
```

La logica di questa routine è scandire tutte le collezioni che fanno parte della gerar-
chia VBIDE.VBI
c'è la variabile AllProjects, che è impostata alla collezione VBProjects per l'istanza VBIDE.VBE. Viene esaminato ogni progetto della collezione VBProjects. La collezione VBComponents di un VBProject rappresenta i moduli di un progetto. Viene esaminato ogni componente della collezione VBComponents. La variabile TheCurrentComponent rappresenta un componente. Se la proprietà HasOpenDesigner di TheCurrentComponent è True, significa che il componente è un contentore in grado di ospitare controlli, come lo sono un form o un modulo controllo d'utente. Inoltre, il suo designer è aperto. Viene passata in rassegna TheCurrentComponent.Designer.VBControls, e a ogni controllo vengono modificate le appropriate proprietà.

Aggiunta del codice per la connessione

Il Listato 29-5 contiene il codice del designer dell'add-in Secrets che gestisce la connessione dell'add-in Change Colors all'ambiente VB.

Listato 29.5 *Connessione di Cbange Colors.*

```
Public FormDisplayed          As Boolean
Public VBlnstance             As VBIDE.VBE
Dim mcbMenuCommandBar         As Office.CommandBarControl
Dim mfrmAddIn                 As New ShowForm
Public WithEvents MenuHandler As CommandBarEvents

'questo metodo aggiunge l'Add-IN a VB

Private Sub AddinInstance_OnConnection(ByVal _
   Application As Object, ByVal ConnectMode As _
   AddInDesignerObjects.ext_ConnectMode, _
   ByVal AddInlnst As Object, custom() As Variant)
   On Error GoTo error_handler

   'salva l'istanza di VB
   Set VBlnstance = Application

   'questo è un buon punto per impostare un'interruzione
   'e verificare oggetti addin, proprietà e metodi

   If ConnectMode = ext_cm_External Then
      'usato dalla barra del wizard per avviare questo wizard
      mfrmAddIn.VBIDE = VBlnstance
   Else
      Set mcbMenuCommandBar = AddToAddInCommandBar _
        ("C&hange Color Add-In")
      'sincronizza l'evento
      Set Me.MenuHandler = _
         VBInstance.Events.CommandBarEvents(mcbMenuCommandBar)
   End If

   If ConnectMode = ext_cm_AfterStartup Then
      If GetSetting(App.Title, "Settings", "DisplayOnConnect", _
         "0"), = "1" Then
      'imposta per visualizzare il form alla connessione
      End If
   End If

Exit Sub

error_handler:
   MsgBox Err.Description
End Sub

'questo metodo elimina l'Add-In da VB

Private Sub AddinInstance_OnDisconnection(ByVal RemoveMode As _
   AddInDesignerObjects.ext_DisconnectMode, custom() As Variant)
   OnErrorResumeNext
```

```
    'elimina la voce dalla barra dei comandi
    mcbMenuCommandBar.Delete

    'chiude l'Add-In
    If FormDisplayed Then
        SaveSetting App.Title, "Settings", "DisplayOnConnect", "1"
        FormDisplayed = False
    Else
        SaveSetting App.Title, "Settings", "DisplayOnConnect", "0"
    End If

    Unload mfrmAddIn
    Set mfrmAddIn = Nothing
End Sub

Private Sub IDTExtensibility_OnStartupComplete _
    (custom() As Variant)
    If GetSetting(App.Title, "Settings", _
        "DisplayOnConnect", "0") = "1" Then
        'imposta per visualizzare il forni alla connessione
    End If
End Sub

Function AddToAddInCommandBar(sCaption As String) As
    Office.CommandBarControl
    Dim cbMenuCommandBar As Office.CommandBarControl
    Dim cbMenu As Object

    On Error GoTo AddToAddInCommandBarErr

    'cerca il menu Add-Ins
    Set cbMenu = VBInstance.CommandBars("Add-Ins")
    If cbMenu Is Nothing Then
        'non è disponibile, fallisce
        Exit Function
    End If

    'lo aggiunge alla barra dei comandi
    Set cbMenuCommandBar = cbMenu.Controls.Add(1)
    'imposta la didascalia
    cbMenuCommandBar.Caption = sCaption

    Set AddToAddInCommandBar = cbMenuCommandBar

    Exit Function

AddToAddInCommandBarErr:

End Function

Private Sub MenuHandler_Click(ByVal CommandBarControl _
    As Object, handled As Boolean, _
        CancelDefault As Boolean)
    mfrmAddIn.VBIDE = VBInstance
    mfrmAddIn.AfterClick
End Sub
```

Onesto funziona in modo molto simile all'esempio di connessione presentato prima in questo capitolo.

Si noti che si può posizionare una voce per il proprio add-in su qualsivoglia menu dì VB:

```
Set cbMenu = VBInstance.CommandBars("Format")
```

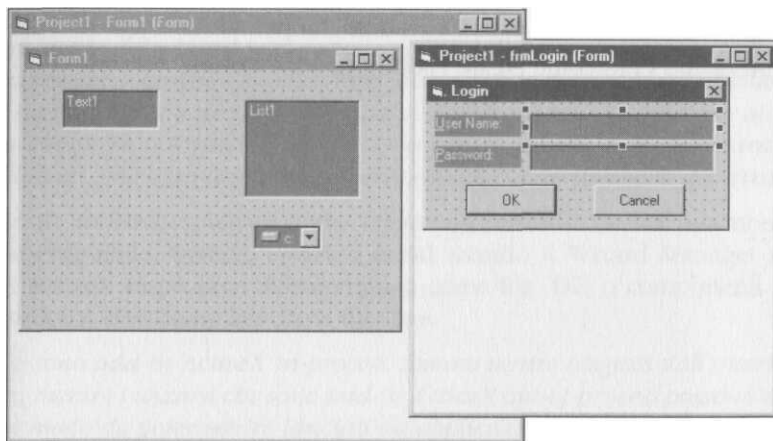
Lo si può posizionare sul menu in qualunque posizione si gradisce:

```
Set cbMenuCommandBar = cbMenu.Controls.Add(before:=3)
```

È anche molto facile includere un separatore prima della voce:

```
cbMenuCommandBar.BeginGroup = True
```

Dopo aver detto tutto ciò, la connessione risulta abbastanza facile da gestire, pur di averne appreso la meccanica. A questo proposito, navigare fra gli oggetti e le collezioni che compongono la gerarchia del mare VBIDE. VBE è piuttosto divertente!
Dopo aver messo a posto il codice per la connessione, aver compilato il proprio add-in, averlo aggiunto a VBaddin.ini, e averlo abilitato nell'Add-In Manager, lo si può usare per andare a modificare molti controlli simultaneamente, come mostrato nella Figura 29.25.

Figura 29.25
Ladd-in Change Colors modifica efficientemente le proprietà di ogni controllo che trova.

Riepilogo

Questo capitolo ha presentato l'affascinante argomento degli add-in (aggiunte). Si tratta di componenti applicativi ActiveX che usano le voci esportate dell'oggetto VBIDE.VBE per modificare l'istanza corrente dell'IDE di Visual Basic. Dopo aver letto il capitolo, si dovrebbero avere le informazioni che servono per creare il proprio sofisticato add-in.

- Abbiamo compreso che cosa sono gli add-in.
- Abbiamo trattato i diversi tipi di add-in.
- Abbiamo visto come usare l'Add-In Manager.
- Abbiamo trattato gli add-in forniti con VB.
- Abbiamo scoperto la barra degli strumenti degli add-in.
- Abbiamo spiegato la gerarchia di oggetti dell'oggetto VBIDE.VBE di Visual Basic.
- Abbiamo visto come accedere e manipolare i membri della gerarchia.
- Abbiamo trattato i metodi Connection e Disconnection.
- Abbiamo visto come creare un semplice add-in.
- Abbiamo trattato la registrazione di componenti.
- Abbiamo scoperto il codice del modulo di classe per la connessione.
- Abbiamo visto come creare voci di menu per add-in.
- Abbiamo visto come creare l'add-in Change Colors.

COSTRUZIONE DI UN WIZARD

- Esecuzione del Wizard Manager
- L'interfaccia utente del Wizard Manager
- Fondamenti dei wizard
- Operazioni con i riquadri dei wizard
- Aggiunta di una bitmap alla voce di menu del wizard

Il Wizard Manager, talvolta bizzarramente chiamato il wizard dei wizard, è un add-in per Visual Basic che facilita la creazione di wizard, che sono a loro volta degli add-in. Per ulteriori informazioni sugli add-in, vedere il Capitolo 29.

L'interfaccia utente dei wizard non è idonea per qualunque situazione. Nelle proprie applicazioni, si dovrebbe usare un 'interfaccia utente di wizard solamente se si è sicuri che sia appropriata. Le interfacce utente di wizard sono ottimali per aiutare gli utenti a compiere operazioni che possono venire intuitivamente separate in argomenti distinti, con ogni argomento corrispondente a un pannello del wizard.

Nel Capitolo 8, abbiamo spiegato come creare un wizard, che sia una normale applicazione eseguibile. Invece, i wizard creati usando il Wizard Manager sono componenti ActiveX *in-process,* cioè compilati come file .Dll, o componenti ActiveX *out-of-process,* cioè compilati come file .Exe.

I *wizard che sono add-in ActiveX in-process devono venire eseguiti dall'interno di Visual Basic, mentre i wizard che sono add-in ActiveX out-of-process possono essere progettati in modo da poter venire lanciati da applicazioni client ActiveX, come le applicazioni di Microsoft Office.*

Questo capitolo tratta l'esecuzione del Wizard Manager per costruire un wizard. Lo scopo primario del Wizard Manager è di aiutare gli sviluppatori a gestire le questioni di visualizzazione dei pannelli. Chi ha creato i propri wizard, o ha seguito l'esempio del Capitolo 8, si sarà reso conto che tali questioni possono essere parecchio complesse, a causa del fatto che i wizard tipicamente sono costituiti da un solo form. Per i wizard creati mediante il Wizard Manager, il nome di questo form è frmWizard.

L'illusione del movimento fra pannelli viene creata manipolando le proprietà Visible dei controlli del form. Oltre ad aiutare con questo gioco di prestigio, il Wizard Manager fornisce il codice modello che si occupa di creare un add-in funzionale. (Si veda il Capitolo 29 per ulteriori informazioni in merito).

Esecuzione del Wizard Manager

Il Wizard Manager è fornito con le edizioni Professional e Enterprise di Visual Basic ma di default non è abilitato. Per abilitare il Wizard Manager di VB6, lo si deve selezionare nell'Add-In Manager, come mostrato nella Figura 30.1. Dopo che il Wizard Manager è stato abilitato, comparirà la relativa voce nel menu *Add-Ins* di Visual Basic, come mostrato nella Figura 30.2.

Figura 30.1 *Si può abilitare il Wizard Manager usando l'Add-In Manager.*

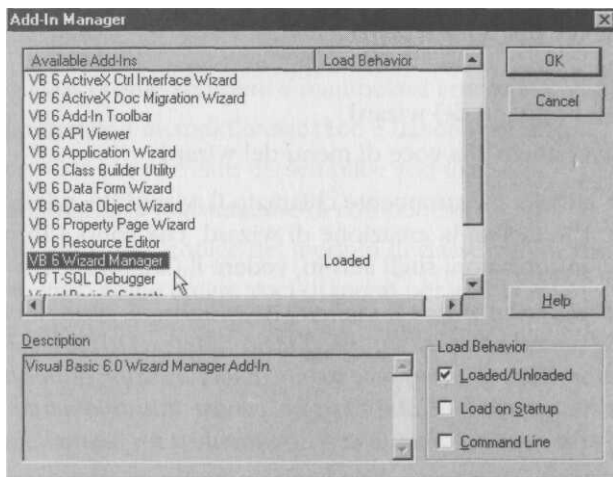



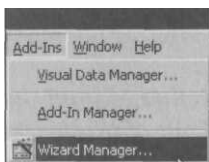
Figura 30.2 *Il Wizard Manager, quando viene abilitato, pone nel menu Add-Ins di VB una voce relativa a sé.*

Per avviare il Wizard Manager, si fa clic sulla voce *Wizard Manager.* A meno che sia già aperto un progetto di wizard, verrà chiesto se si vuole creare un nuovo progetto (vedere la Figura 30.3).

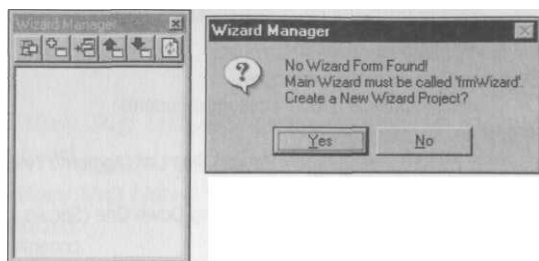
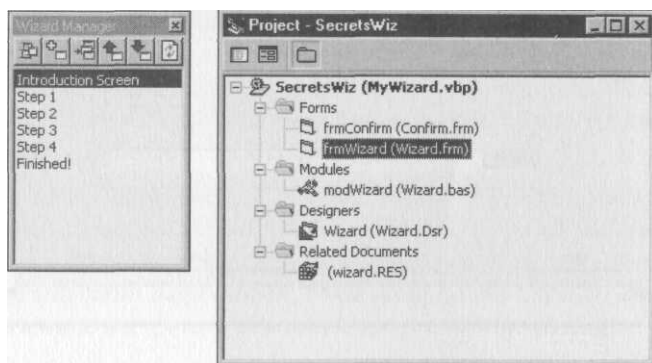
Figura 30.3
Il Wizard manager si offre di creare un nuovo progetto di Wizard.

Il progetto creato dal Wizard Manager contiene il form del wizard, di nome `frmWizard`, nonché un modulo di codice, un modulo di classe per gestire la connettività dell'add-in, e un file di risorse (Wizard.Res) per immagazzinare le stringhe e le bitmap che vengono usate per personalizzare il wizard. La Figura 30.4 mostra un modello di progetto di wizard nel Project Explorer, insieme all'interfaccia utente del Wizard Manager.

Figura 30.4
Un progetto modello di wizard contiene un form di wizard, moduli di codice e di classe e un file di risorse

L'interfaccia del Wizard Manager

L'interfaccia utente del Wizard Manager ha due scopi. In primo luogo, elenca tutti i pannelli del progetto del wizard, come mostrato nella Figura 30.5. Ogni pannello, quando viene selezionato, compare nel designer di `frmWizard`, come si vede nella Figura 30.6. Come si può notare nella Figura 30.6, quando si usa il Wizard Manager per aprire un pannello, c'è la possibilità di modificare il nome del passo rappresentato dal pannello.

Figura 30.5 *L'interfaccia del Wizard Manager aiuta a navigare fra i pannelli.*

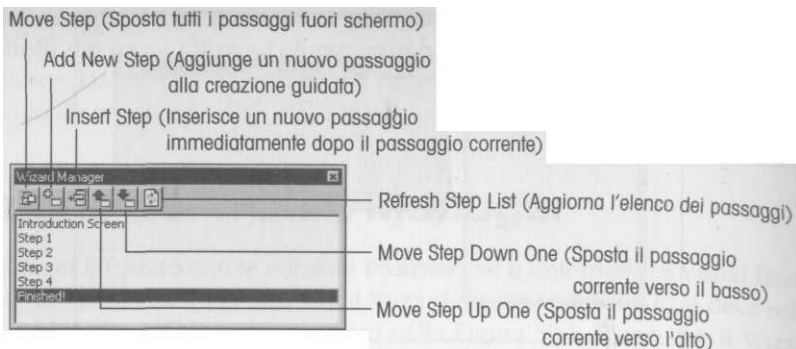
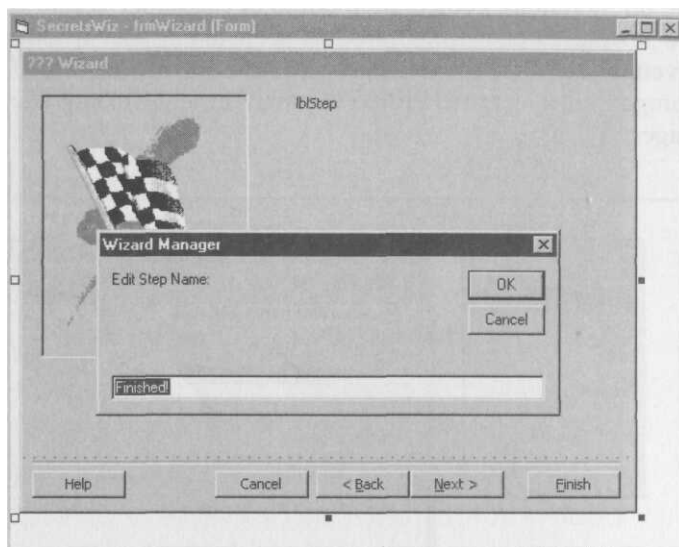



Figura 30.6 *La selezione di un pannello di wizard nel Wizard Manager lo fa visualizzare nel designer di frm Wizard.*

In realtà, i pannelli di wizard non sono affatto pannelli. Il form frmWizard *in effetti contiene una matrice di riquadri (*frame). Il *"pannello" virtuale attuale del wizard è il membro di quella matrice in posizione tale sul form da essere visibile. Ogni pannello è indicato anche come* passo *del wizard.*

Il Wizard Manager fornisce sei pulsanti di toolbar (Figura 30.5):

- Il pulsante *Move Step* sposta il pannello di wizard attualmente visibile fuori dall'area di schermo visibile del wizard. Abitualmente, si dovrebbe fare clic su questo pulsante dopo aver completato il proprio wizard; altrimenti, l'ultimo pannello su cui si è lavorato, tipicamente l'ultimo del wizard, apparirà per primo quando si avvia il wizard.

- Il pulsante *Add Step* aggiunge un nuovo pannello virtuale al wizard, prima del pannello "Finished!", ma dopo tutti gli altri passi.
- Il pulsante *Insert Step* crea un nuovo passo prima del passo attualmente selezionato.
- Il pulsante *Move Step Up One* sposta il pannello selezionato in su di un passo nel wizard.
- Il pulsante *Move Step Down One* sposta il pannello selezionato in giù di un passo nel wizard.
- *Refresh Step List* aggiorna l'elenco dei passi tenendo conto delle modifiche apportate.

Fondamenti dei wizard

Non è necessario comprendere molto del progetto creato dal Wizard Manager, né di come il Wizard Manager funziona in generale, per creare un wizard personalizzato funzionante.

Utilizzo del file di risorse

La maggior parte delle stringhe visualizzate dal wizard vengono caricate dal file di risorse che accompagna il progetto. Questo file viene creato dal Wizard Manager con il nome Wizard.Res. Si troveranno ulteriori informazioni sull'utilizzo delle risorse esterne nel Capitolo 16 e nel Capitolo 18.

Per modificare i file di risorse, si può usare il Resource Editar di VB. Il Resource Editar di VB è, esso stesso, un add-in di Visual Basic. Prima di poterlo usare, lo si deve caricare nell'Add-In Manager. Dopo che è stato caricato, può venire lanciato dalla barra degli strumenti degli add-in, come spiegato nel Capitolo 29. Il Resource Editor di VB è orientato ad operare con un elenco prestabilito di lingue.

Il Resource Editor fornito come parte di Visual C++ 6.0 (incluso in Visual Studio) è più potente e flessibile dell'add-in di Visual Basic. Si può usare Visual C++ per modificare la *String Table* (tabella di stringhe) contenuta in Wizard.Res, come mostrato nella Figura 30.7. Per ritrovare nel proprio wizard personalizzato le modifiche apportate alle stringhe, occorre compilarlo e lanciarlo (vedere la Figura 30.8).

Figura 30.7

Si può usare VisualC++ per modificare la String Table in Wizard.Res.

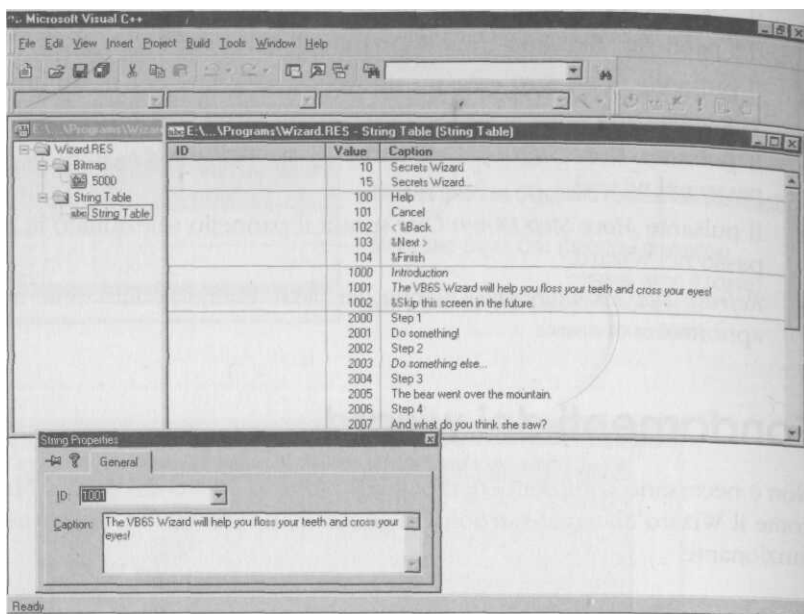
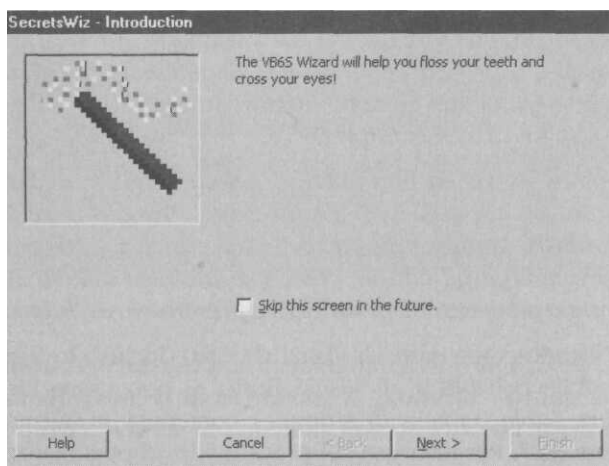



Figura 30.8

I *valori introdotti nella String Table del file di risorse vengono visualizzati dal Wizard dopo che il file è stato compilato e lanciato.*

Recupero dei valori della tabella delle stringhe

Il wizard utilizza del codice davvero brillante per recuperare correttamente i valori della tabella delle stringhe e assegnarli alle giuste proprietà dei controlli. Ecco come funziona. Ogni controllo che ha una stringa corrispondente nella tabella delle stringhe ha la sua proprietà Tag impostata all'ID della stringa. Per ulteriori informazioni sull'utilizzo della proprietà Tag di un controllo, vedere il Capitolo 8. Per esempio, un controllo etichetta del passo *Introduction Screen* del wizard ha la sua proprietà Tag impostata a 1001. La proprietà Tag 1001 è l'ID della stringa "The VB6S Wizard

will help you floss your teeth and cross your eyes!". Perciò, questa stringa viene caricata nell'etichetta prima che il wizard venga mostrato.
Una routine di nome LoadResStrings nel modulo ModWizard.Bas gestisce l'effettivo lavoro di porre le stringhe nei controlli che hanno valori di Tag. LoadResStrings viene chiamata dall'evento Load di frmWizard, con il form passato come argomento. Come si può vedere nel Listato 30.1, LoadResStrings tenta di caricare tutte le stringhe di testo del form del wizard e dei suoi controlli, dove la proprietà del Tag del controllo corrispondente ha un valore numerico. Siccome le stringhe dei menu non hanno una proprietà Tag, esse vengono caricate in base agli ID immagazzinati nelle loro etichette.

Listato 30.1 *Caricamento delle stringhe dei controlli e dei menu.*

```
Sub LoadResStrings(frm As Form)
   On Error Resume Next
   Dim ctl As Control
   Dim obj As Object
   'imposta la didascalia del form
   If IsNumeric(frm.Tag) Then
      frm.Caption = LoadResString(CInt(frm.Tag))
   End If

   'imposta le didascalie dei controlli con la proprietà
   'Caption per le voci di menu e la proprietà
   'tag per tutti gli altri controlli
   For Each ctl In frm.Controls
      If TypeName(ctl) = "Menu" Then
         If IsNumeric(ctl.Caption) Then
            If Err = 0 Then
               ctl.Caption = LoadResString(CInt(ctl.Caption))
            Else
               Err = 0
            End If
         End If
      Elself TypeName(ctl) = "TabStrip" Then
         For Each obj In ctl.Tabs
            If IsNumeric(obj.Tag) Then
               obj.Caption = LoadResString(CInt(obj.Tag))
            End If
            'check for a tooltip
            If IsNumeric(obj.ToolTipText) Then
               If Err = 0 Then
                  obj.ToolTipText =
                     LoadResString(CInt(obj.ToolTipText))
               Else
                  Err = 0
               End If
            End If
         Next
      Elself TypeName(ctl) = "Toolbar" Then
         For Each obj In ctl.Buttons
            If IsNumeric(obj.Tag) Then
               obj.ToolTipText = LoadResString(CInt(obj.Tag))
```

```
            End If
         Next
      ElseIf TypeName(ctl) = "ListView" Then
         For Each obj In ctl.ColumnHeaders
            If IsNumeric(obj.Tag) Then
               obj.Text = LoadResString(CInt(obj.Tag))
            End If
         Next
      Else
         If IsNumeric(ctl.Tag) Then
            If Err = 0 Then
               ctl.Caption = LoadResString(CInt(ctl.Tag))
            Else
               Err = 0
            End If
         End If
         'check for a tooltip
         If IsNumeric(ctl.ToolTipText) Then
            If Err = 0 Then
               ctl.ToolTipText=_
                  LoadResString(CInt(ctl.ToolTipText))
             Else
                Err = 0
             End If
         End If
      End If
   Next
End Sub
```

Personalizzazione del wizard

Per personalizzare il proprio wizard, bisogna:

- Modificare i valori della tabella delle stringhe in Wizard.Res adattandoli ai bisogni del proprio wizard, come descritto prima.
- Utilizzare il Wizard Manager per aggiungere, eliminare, o riordinare i pannelli del wizard.
- Utilizzare il Wizard Manager per visualizzare diversi pannelli del wizard. Per ogni pannello, aggiungere controlli e regolare le proprietà dei controlli secondo necessità. Assicurarsi di aggiungere gli ID delle stringhe alle proprietà Tag dei nuovi controlli e le corrispondenti voci della tabella delle stringhe, come descritto prima.
- Adeguare le costanti del modulo di codice Wizard.Bas alle esigenze del proprio wizard. Per esempio:

```
Global Const WIZARD NAME = "Secrets Wizard"
```

- Aggiungere del codice all'evento Click di `cmdNav`, nel modulo `frmWizard`, in modo da realizzare lo scopo del wizard quando l'utente fa clic sul pulsante Finish *realizzare.* La posizione in cui porre questo codice è indicata da un commento:

```
Case BTN_FINISH
    'qui va il codice di creazione del wizard
```

Trasformazione in add-in

prima che si possa usare il wizard creato, si devono seguire i passi necessari per compilarlo e registrarlo come add-in. Si veda il Capitolo 29 per avere informazioni dettagliate su questi passi. Si potrebbe voler modificare la posizione della voce di menu di Visual Basic che viene aggiunta per il Wizard, eventualmente spostandola dal menu *Add-Ins.* In tal caso, il codice da modificare è nella routine AddToAddIn-CommandBar.
È una buona regola personalizzare il nome e la descrizione usati dall'Add-In Manager per il wizard. Per farlo, si può usare l'interfaccia utente del designer di wizard, come mostrato nella Figura 30.9. Il nome che il Wizard Manager ha dato a questo modulo di classe è Wizard.Dsr.

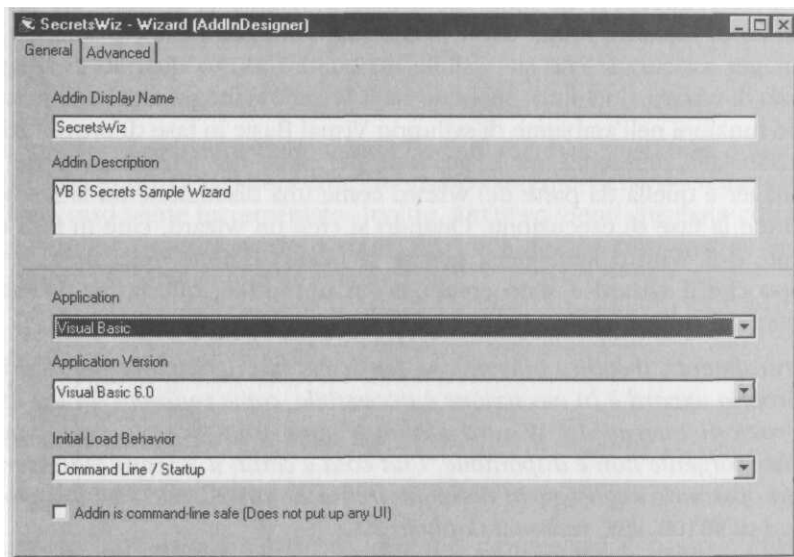
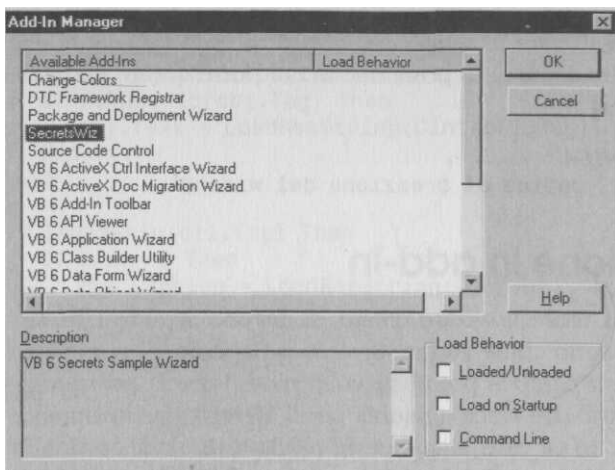
Figura 30.9 *Si può usare il designer di wizard per modificare il nome e la descrizione del wizard che appaiono nell'Add-In Manager.*

Il progetto di wizard deve venire compilato come componente ActiveX. Questo significa che l'opzione *Project Type* deve essere impostata a *ActiveXDLL* o a *ActiveX EXE* nella scheda *General* della finestra di dialogo *Project Properties.* Dopo che il Wizard è stato compilato, il nuovo nome e la nuova descrizione compaiono nell'Add-In Manager, come mostrato nella Figura 30.10.

Figura 30.10 *Le modifiche apportate al nome e alla descrizione del Wizard si riflettono nell'Add-In Manager.*

Incorniciato di nuovo

Ogni passo del wizard è in realtà un riquadro *(frame),* che fa parte della matrice di riquadri `fraStep()`. In ogni momento, è visibile solamente un membro della matrice di riquadri. Prima che il wizard sia stato compilato, è compito del Wizard Manager assicurarsi che sia visibile il riquadro giusto quando ci si sposta da un passo di wizard a un altro. Sebbene sia il Wizard Manager che l'add-in wizard generano funzioni nell'ambiente di sviluppo Visual Basic in fase di progettazione, si può pensare alla differenza tra la gestione dei passi del wizard da parte del Wizard Manager e quella da parte del wizard come una distinzione tra la fase di progettazione e la fase di esecuzione. Quando si crea un wizard, cioè in fase di progettazione, è il Wizard Manager a gestire la manipolazione della matrice di riquadri. Dopo che il wizard è stato compilato in un add-in, questa manipolazione viene gestita dal codice interno del wizard stesso.

Naturalmente, il codice interno che gestisce la manipolazione dei riquadri quando il proprio wizard è in esecuzione è accessibile, come vedremo fra un attimo; ma le operazioni interne del Wizard Manager sono una "scatola nera", perché il suo codice sorgente non è disponibile. Una cosa è certa: il Wizard Manager, ciò che fa, lo fa operando sugli oggetti della gerarchia di `VBIDE.VBE`. *Per maggiori informazioni su* `VBIDE.VBE`, *vedere il Capitolo 29.*

Dopo che il wizard è stato compilato in un componente ActiveX, la manipolazione della matrice di riquadri è gestita da una routine di nome `SetStep`, chiamata dall'evento Click di `cmdNav`. Questo codice è incluso nel proprio wizard! L'evento Click di `cmdNav` scatta ogni volta che l'utente fa clic su uno dei pulsanti del wizard. Ecco l'intera routine:

```
Private Sub cmdNav_Click(Index As Integer)
    Dim nAltStep As Integer
    Dim lHelpTopic As Long
```

```
Dim rc As Long

Select Case Index
   Case BTN_HELP
      mbHelpStarted = True
      lHelpTopic = HELP_BASE + 10 * (1 + mnCurStep)
      rc=WinHelp(Me.hwnd,HELP_FILE,_
         HELP_CONTEXT, IHelpTopic)
   Case BTN_CANCEL
      Unload Me
   Case BTN_BACK
      'mettere qui i casi speciali
      'per saltare a passi alternativi
      nAltStep = mnCurStep - 1
      SetStep nAltStep, DIR_BACK
   Case BTN_NEXT
      'mettere qui i casi speciali
      'per saltare a passi alternativi
      nAltStep = mnCurStep + 1
      SetStep nAltStep, DIR_NEXT
   Case BTN_FINISH
      'qui va il codice di creazione del Wizard
      Unload Me
      If GetSetting(APP_CATEGORY, WIZARD_NAME, _
         CONFIRM_KEY, vbNullString) = vbNullString Then
         frmConfirm.Show vbModal
      End If
EndSelect
End Sub
```

L'unica differenza fra la pressione del pulsante *Back* (indietro) e quella del pulsante *Next* (avanti) è che in un caso il contatore del passo corrente viene decrementato, mentre nell'altro caso viene incrementato. Inoltre, SetStep viene chiamata con un argomento che indica la direzione del wizard. Ecco la porzione della routine Set-Step che gestisce lo spostamento a un nuovo passo:

```
'spostamento a un nuovo passo
fraStep(mnCurStep).Enabled=False
fraStep(nStep).Left = 0
If nStep <> mnCurStep Then
   fraStep(mnCurStep).Left = -10000
End If
fraStep(nStep).Enabled = True
```

Questo è davvero impressionante! Il riquadro che rappresenta il passo corrente viene posto nell'area visibile impostando a 0 la sua proprietà Left. Tutte gli altri riquadri vengono spostati "fuori scena" impostando a -10000 la loro proprietà Left, ben fuori dalle porzioni visibili del form del wizard. Lo si può pensare come un procedimento in cui i passi che non sono in uso sono tenuti in una specie di ripostiglio. Questo ripostiglio è fuori dalla vista, e, come si suoi dire, "lontano dagli occhi, lontano dal cuore".

Le coordinate delle dimensioni e della posizione del form sono espresse in twip. Per ulteriori informazioni sulle unità di misura e sulla manipolazione dello schermo, si veda il Capitolo 19. Sono in twip anche le dimensioni usate per manipolare la matrice di riquadri usata nel codice del wizard.

Qualunque posizione che sia minore di zero non sarà visibile agli utenti. Ogni elemento della matrice di riquadri è largo 7155 twip (come indicato dalla proprietà Width del controllo). Ciò significa che, quando la proprietà Left dell'elemento del riquadro è impostata a -10,000, il suo bordo destro verrà posizionato a -2845 twip, ben fuori dallo schermo visibile. Immaginatevi una catasta di riquadri di wizard sovrapposti, tutti a sinistra del riquadro visibile, e tutti invisibili all'utente

.

Avrete forse notato che è stata aggiunta un'icona, o una bitmap, alla voce di menu del wizard Secrets, come mostrato nella Figura 30.11. Questo è stato fatto nella routine OnConnection di AddInInstance, che fa parte del modulo designer di wizard. Le routine per la connessione e per la disconnessione degli add-in funzionano allo stesso modo di un add-in non di tipo wizard, descritto nel Capitolo 29.

Figura 30.11
Una bitmap [s]tata aggiunta [alla] voce di menu [del w]izard Secrets.

Il terzo parametro di AddToAddInCommandBar carica una bitmap dal file delle risorse esterne, insieme al testo che appare nel menu. Tale parametro è facoltativo, e la stessa sintassi funziona per qualunque menu, non solo per il menu *Add-Ins:*

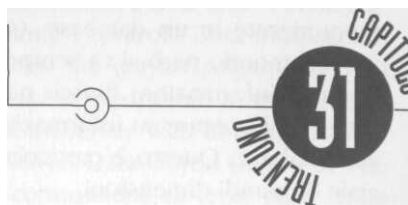
```
Set mcbMenuCommandBar = AddToAddInCommandBar _
    (VBInstance, LoadResString(15), LoadResPicture(5000, 0))
```

Riepilogo

Come si può non trovare simpatica l'espressione "wizard dei wizard"? Questo capitolo ha spiegato come usare il wizard dei wizard, cioè il Wizard Manager. Con qualunque nome lo si chiami, il Wizard Manager è un add-in che svolge le funzioni di impresario, sempre pronto ad aiutarvi a creare i vostri add-in di tipo wizard.

- Abbiamo trattato l'esecuzione del Wizard Manager.
- È stato spiegato come utilizzare l'interfaccia utente del Wizard Manager.
- È stato spiegato come caricare le stringhe di testo di un wizard da un file di risorse esterno.
- Abbiamo visto la corrispondenza fra le proprietà Tag dei controlli di un wizard e gli ID della sua tabella delle stringhe.
- Abbiamo visto come personalizzare il wizard modello.
- Abbiamo trattato la compilazione di un wizard come add-in.
- Abbiamo scoperto come il wizard manipola i suoi riquadri.
- Abbiamo trattato l'aggiunta di una bitmap alla voce di menu di un wizard.

DATABASE, INSTALLAZIONE E GUIDA

I CONCETTI FONDAMENTALI DEI DATABASE

- La vita è un database
- Architettura multilivello (multitier)
- Server di database
- Sistemi di gestione dei database relazionali
- Database e OOP
- SQL
- Controllo Data di Visual Basic
- Controlli sensibili ai dati

Nel mondo reale molti programmi, qualunque sia la loro complessità, utilizzano i database. Appena cominciate a gestire delle cose, che siano persone, soldi, materiali, impostazioni del software e così via, avete bisogna di un database. Nella pratica, non ha senso reinventare la ruota: i programmi che hanno bisogno delle funzionalità dei database interagiscono con il software per i database esistente. In particolare, quando viene usato Visual Basic per risolvere dei problemi di questo tipo, i programmatori VB spesso creano delle applicazioni che non sono niente di più (e niente di meno) che delle interfacce per i database.
I database sono come le tubature o i cavi elettrici nascosti nei muri di casa vostra. Nessuno ci pensa fino a quando non se ne ha bisogno, ma a quel punto diventa importante sapere come funzionano. Questo capitolo spiega i concetti di cui avrete bisogno in modo che possiate creare con successo dei programmi che lavorano con i database.

La vita è un database

Nel Capitolo 14 ho mostrato che quasi tutti i problemi di programmazione possono essere risolti con un approccio orientato agli oggetti, e che ogni cosa, compresi per esempio i sistemi per le previsioni metereologiche e i sistemi per la borsa, può essere vista come collezione di oggetti.

Invertendo questo concetto, tutti gli oggetti possono essere visti come parte di database. In questo senso, la vita stessa è un database. Più avanti in questo capitolo, in "Database e OOP", porterò questo concetto alla sua logica conclusione, e creeremo un software per i database nel quale i costrutti del database potranno comportarsi in un modo orientato agli oggetti.
Più prosaicamente, quasi tutte le informazioni importanti per una persona sono memorizzate in un database. (Ovviamente, prima dei computer, questo non era vero. Il mondo però si va sempre più informatizzando e una quantità sempre maggiore di informazioni unisce nei database.) Il compito di molti programmi complessi è di operare su informazioni importanti per la gente, il che significa operare sui database. Questo è particolarmente vero in un contesto aziendale o commerciale di grandi dimensioni.
Uno degli scopi primari della versione 6 di Visual Basic è di essere uno strumento per creare applicazioni aziendali. In una applicazione per l'azienda, la funzione della porzione di Visual Basic sarà principalmente di *front-end,* o interfaccia utente

Architettura multilivello

L'architettura multilivello o *multitier,* detta anche *client/server,* indica la divisione strutturale delle applicazioni in moduli incapsulati, ognuno dei quali ha una particolare funzionalità e interfacce ben definite. Per ulteriori informazioni su questo argomento andate a vedere la sezione dedicata alla progettazione delle applicazioni nel Capitolo 13.
Le applicazioni database tipiche sono applicazioni client/server a due oppure a tre livelli. Il modello a due livelli indica che una applicazione client, con la propria logica e con le proprie procedure, comunica con un server di database (il quale, oltre a contenere le informazioni o i dati, può contenere della logica, per esempio delle procedure memorizzate).
Nel modello a tre livelli, viene aggiunto un livello intermedio, alcune volte chiamato *deposito* o *repository,* tra il client ed il server. Il client gestisce l'interfaccia utente e talvolta include il codice che realizza le "regole aziendali". Il livello intermedio passa le transazioni generate dal client al server. Il server fa quello che deve fare con la transazione e invoca il livello intermedio per mandare una risposta al client.

Server di database

Mentre siamo sull'argomento dei server, bisogna notare che esistono molti tipi di server diversi. I server con cui avete probabilmente lavorato sono i server di rete, i server Web e le applicazioni server ActiveX (vedere il Capitolo 23).
Quello che questi server e i server di database hanno in comune è che rispondono alle richieste e che restituiscono qualcosa a chi ha inviato la richiesta. Un server deve anche essere eseguito in background (server di rete, server dei database aziendali, server Web), oppure deve avere dei metodi per l'attivazione esterna (componenti ActiveX).

Generalmente, ma non sempre, le richieste fatte a un server di database prendono la forma di istruzioni SQL relative a specifiche tabelle nel database (vedere la sezione "SQL" più avanti in questo capitolo). In ogni modo, un server di database d'impresa può gestire più di un singolo database.
Nel contesto di una soluzione di una applicazione Visual Basic per l'azienda, il server di database generalmente non interagisce con il client VB da solo. Potete collegare direttamente i controlli a un database mediante i controlli Data intrinseci di VB come verrà spiegato più avanti questo capitolo. Ma questo solitamente non rende le vostre applicazioni sufficientemente robuste o espandibili in un contesto aziendale. Alcune altre soluzioni sono l'impiego di un server Web in congiunzione con il server di database, mediante la tecnologia ActiveX Data Object (ADO) di VB6 (vedere il capitolo 32), oppure con strumenti di connessione di terze parti come Data Director for VB (DDVB) di Informix.

Sistemi di gestione dei database relazionali

I primi database sono stati costruiti usando quello che ora è definito modello *gerarchico.* In un database gerarchico è difficile modificare le colonne in una tabella una volta che queste sono state create.
Il modello del primo sistema di gestione dei database relazionale, solitamente detto *rdbms, è* stato formulato dal ricercatore dell'IBM E. F. Codd nel 1970. Il primo prodotto commerciale basato su quel concetto non fu dell'IBM, ma invece di una piccola società chiamata Oracle.
I database relazionali hanno una serie di caratteristiche che li distingue dai database gerarchici, ma la più importante è la capacità di modificare la struttura del database al volo. Purché non eliminiate dei dati da cui dipende l'applicazione, questo significa che le applicazioni potranno ancora funzionare anche dopo che sono state fatte delle modifiche. Un motivo per cui gli RDBMS sono così flessibili è che i dati sono archiviati in tabelle che sono in gran parte indipendenti le une dalle altre.
Ecco alcuni importanti termini e concetti degli RDBMS:

- **Vista.** Una vista è una porzione (per esempio, delle colonne o delle righe) di una o più tabelle. Le viste sono usate per isolare i dati che interessano.
- **Schema.** Uno schema del database è la struttura dell'organizzazione, generale del database: le tabelle, come sono in relazione tra di loro, le colonne nelle tabelle e quale tipo di dati contengono.
- **Dominio.** L'insieme di tutti i valori in un attributo di una relazione, per esempio una colonna in una tabella è il dominio dell'attributo. Se il mio articolo è disponibile nei colori rosso, bianco o blu, e la mia tabella *Prodotti* ha una colonna *ColoreArticolo,* allora il dominio di *ColoreArticolo* è l'insieme dei tre colori rosso, bianco e blu.

- **Vincoli.** I vincoli sono regole applicate ai membri del database, solitamente alle colonne. Per esempio, se una colonna è vincolata come UNIQUE, allora non ci possono essere due elementi identici nella colonna Tentare di aggiungere un secondo elemento identico nella colonna provocherà un errore del server di database.

Database e OOP

Per la sua natura, un database è una "cosa" rigida, gerarchica. È un sistema definito da relazioni statiche. Se la vita è un database, come abbiamo proclamato all'inizio di questo capitolo, e se la programmazione orientata agli oggetti è il metodo migliore per risolvere parecchi problemi di larga scala, come possono essere riconciliati i database e OOP?

OOP nel contesto di un database coinvolge due diversi problemi:

1. *I database normalmente memorizzano solo tipi standard di variabili, come numeri e stringhe. Cosa dobbiamo fare per tutti gli altri tipi di oggetti che vogliamo poter gestire in un database? Con il successo di Internet, questo è diventato un problema molto serio: filmati, immagini, suoni, pagine HTML e molti altri tipi di oggetti possono spesso essere memorizzati nelle tabelle di un database relazionale convenzionale. Ma la gestione di questi tipi di oggetti è spesso un problema serio.*
2. *Molti dei vantaggi offerti dalla OOP provengono dalla sua abilità di imitare i processi tipici della vita, come l'ereditarietà. Quando un oggetto eredita da un altro oggetto, non dovete partire da zero con il nuovo oggetto. Come possiamo implementare le caratteristiche di OOP come l'ereditarietà nel contesto dei database?*

In risposta al primo problema, sono stati sviluppati dei database che gestiscono gli oggetti invece che variabili standard. Un esempio importante è PostgreSQL, creato alla University of California di Berkeley. Ma i database orientati agli oggetti non sono ancora arrivati nel mondo reale perché a questi database mancano le caratteristiche tipiche dei prodotti aziendali che garantiscano la sicurezza, la scalabilità e le prestazioni.
Michael Stonebraker dell'Università della California a Berkeley, che ha lavorato con Illustra e con InForm1x Software, ha proposto la creazione di un ibrido, il database relazionale ad oggetti, o *ORDBMS.* Un database relazionale ad oggetti implementa i concetti relazionali tradizionali, come l'accesso SQL. Inoltre, ogni tipo di oggetto può essere archiviato nel database come tipo *opaco.*

In un database di Access, si possono archiviare immagini e alcuni altri tipi di oggetti binari come oggetti OLE.

L'implementazione di InForm1x del concetto ORDBMS è chiamata Universal Data Option (UDO). Oltre ad abilitare la possibilità di archiviare ogni tipo di oggetto nel database, UDO è estendibile attraverso una specie di plug-in noto come un *datablade.*

Infine si possono definire tipi di riga che contengono gli attributi per colonne multiple. Questi tipi di riga possono essere usati come base dell'ereditarietà orientata agli oggetti.

SQL

SQL è l'abbreviazione di "Structured Query Language". Creato dai ricercatori dell'IBM nel 1970, insieme con il primo DBMS relazionale, SQL è usato per comunicare con i database relazionali (RDBMS).
Mentre ogni prodotto RDBMS importante, per esempio InForm1x, SQL Server di Microsoft e Oracle, parla un diverso dialetto SQL, i comandi principali di SQL che fanno parte dello standard ANSI SQL sono compresi da tutti i principali RDBMS. SQL può essere diviso in quattro parti, talvolta chiamate *sottolinguaggi:*

- Linguaggio di definizione dei dati (Data Definition Language - DDL)
- Linguaggio di manipolazione dei dati (Data Manipulation Language - DML)
- Linguaggio di amministrazione del sistema (System Administration Language - SAL)
- Linguaggio di query (Query Language)

DDL è usato per creare, modificare e per distruggere le tabelle e gli indici all'interno del database. DML è usato per inserire, modificare e per cancellare le righe della tabella. SAL è usato per gestire il sistema, per esempio per aggiungere la sicurezza o gli schemi di autorizzazione. È il sottolinguaggio di SQL meno standardizzato tra i diversi RDBMS. Il sottolinguaggio Query Language è quello a cui si pensa nella maggior parte dei casi in cui si parla di "SQL".

Per convenzione, le parole chiave di SQL sono in lettere maiuscole, per esempio SELECT.

Principalmente, SQL esiste per fare delle domande al database. Non è un linguaggio procedurale come Visual Basic (oppure come il C o come qualsiasi altro linguaggio di programmazione). Non ha istruzioni per i cicli o per il controllo del flusso.

L'attuale implementazione ANSI SQL è SQL2. SQL3, ancora in fase di sviluppo, incorporerà, sia pure in modo limitato, caratteristiche dei linguaggi procedurali.

Le interrogazioni SQL comprendono generalmente tre parti:

- Cosa (l'istruzione Select, elenca le colonne)
- Da (l'istruzione From, indica le tabelle)
- Dove (l'istruzione Where, fornisce le condizioni logiche)

Per esempio

```
SELECT PartNum, PartDesc FROM Inventory
   WHERE PartPrice > 10.00
```

restituisce il numero del pezzo e la descrizione del pezzo per tutt i pezzi il cui prezzo è maggiore di 10. La clausola WHERE in una istruzione SQL è opzionale e si usa un asterisco (*) per indicare di restituire qualsiasi cosa. Quindi, come esempio, la riga seguente restituisce tutte le righe e le colonne della tabella Inventory:

```
SELECT * FROM Inventory
```

Poiché SQL non è procedurale e molte applicazioni richiedono la logica procedurale, non vedrete applicazioni SQL isolate. Le istruzioni SQL sono solitamente incorporate all'interno di programmi creati in altri linguaggi, come Visual Basic.

Potete aggiungere facilmente dei controlli, ed alcune righe di codice, a un form di VB per illustrare l'esecuzione di istruzioni SQL dall'interno di Visual Basic. (Il progetto d'esempio si trova sul CD-ROM allegato come SQL. Vbp.)

Per vedere come questo funziona in pratica, dovete impostare un form con alcuni controlli. Aggiungete una casella di testo a più righe, un pulsante di comando, un controllo DBGrid, e un controllo Data al form.

Troverete il controllo DBGrid *elencato nella finestra di dialogo* Components *come* Microsoft Data Bound Grind Control. Il *controllo* Data, *trattato nella prossima sezione di questo capitolo, è un controllo intrinseco di VB, e si trova nella vostra Toolbox.*

Il controllo Data è usato per collegare la griglia con un particolare database. Per impostare questa connessione, prima impostate il DatabaseName del controllo Data al database. Nell'esempio, ho usato Nwind.mdb, il database di Access d'esempio che viene distribuito con VB6.

Potete anche impostare la tabella usando la proprietà RecordResource *del controllo* Data. *Non è una cosa indispensabile, ma lo si può fare nell'istruzione SQL che si immetterà.*

Poi, collegate il DBGrid al controllo Data impostando la proprietà DataSource di DBGrid a Data1. Infine, implementate l'esecuzione della SQL immessa nella casella di testo per il database Nwind aggiungendo il seguente codice all'evento Click del pulsante di comando:

```
Data1.RecordSource = Text1.Text
Data1.Refresh
```

Se eseguite questo progetto, potete immettere una istruzione SQL nella casella di testo ed interrogare il database Nwind. Per esempio, potete chiedere il nome ed il cognome di tutti gli impiegati il cui cognome inizia con una lettera uguale a D o con una lettera che si trova dopo la D in ordine alfabetico, ordinati per cognome:

```
SELECT FirstName, LastName
FROM Employees
WHERE LastName > 'D'
ORDER BY LastName
```

Il risultato di questa interrogazione è mostrato nella Figura 31.1.

Figura 31.1 *In Visual Basic potete facilmente eseguire le interrogazioni SQL.*

Spesso vorrete generare delle istruzioni SQL usando le funzioni di manipolazione delle stringhe di VB, invece che lasciarle generare all'utente. Il valore assegnato alla proprietà RecordSource *del controllo* Data *in tal caso sarà creato nel codice, non preso da una casella di testo.*

Il controllo Data di Visual Basic

Il controllo Data è intrinseco a VB6, il che significa che apparirà sempre nella vostra Toolbox. Non dovrete selezionare niente nella finestra di dialogo dei componenti per abilitarlo. Come ho mostrato nella precedente sezione, viene usato nella applicazioni Visual Basic a due livelli per collegare i controlli alla sorgente dei dati.
Tutto quelle che dovete fare per usarlo è impostare la proprietà DatabaseName a un database e la sua proprietà RecordSource a una tabella oppure ad una vista. Una volta che questo è stato fatto, il controllo Data cicla attraverso i valori del suo RecordSource.
Molti controlli di Visual Basic intrinseci possono essere collegati ad una sorgente di dati mediante il controllo Data. Una volta che il controllo Data è stato impostato, assegnate la proprietà DataSource del controllo a cui volete collegarlo al controllo Data (appare nella casella di riepilogo a discesa della finestra *Properties).*

Controlli sensibili ai dati

I controlli intrinseci che sono sensibili ai dati, cioè che possono essere collegati, sono Text Box, List Box, Check Box, Combo Box e Label. (Nel caso di Label, *è* naturalmente di sola lettura.) I controlli sensibili ai dati hanno quattro proprietà:

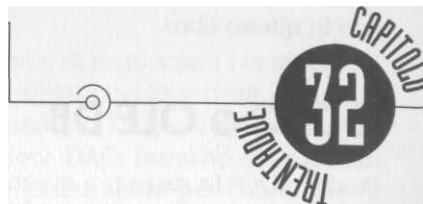
- **DataField,** che collega il controllo ad una colonna
- **DataFormat,** che è usata per formattare i contenuti del controllo collegato al database

- **DataMember,** che è usata per scegliere quale insieme di dati collegare, se esistono più insiemi
- **DataSource,** che è impostato al controllo Data usato per collegare il controllo

Riepilogo

I database governano il mondo. Certamente, se siete sviluppatori professionisti dovrete spendere del tempo lavorando con i database. Questo capitolo ha trattato i concetti base di cui avrete bisogno per avere successo.

- Avete imparato come concettualizzare i database e l'architettura multilivello.
- Ho spiegato sotto quali aspetti un server di database è simile, e in che cosa differisce, rispetto agli altri tipi di server.
- Avete imparato che cosa sono i database relazionali e come OOP interagisce con essi.
- Ho spiegato i concetti base di SQL e come richiamare SQL da una casella di testo di Visual Basic.
- Ho mostrato come funzionano i controlli Data, e come collegare i controlli sensibili ai dati ad una sorgente di dati.

ACTIVEX DATA OBJECT

- Dai Data Access Object (DAO) agli ActiveX Data Object (ADO)
- ODBC e OLE DB
- Uso di DAO per lavorare con i database
- Che cosa sono gli ActiveX Data Object
- Il controllo Data ADO
- Data Environment
- Il controllo DataRepeater

Data Access Object (DAO) è una interfaccia di accesso ai dati: immaginatela come un livello di astrazione orientato agli oggetti, che può essere usato per manipolare i database in VB usando il motore per i database Microsoft Jet (che viene distribuito con VB) oppure ODBC (Open DataBase Connectivity). ActiveX Data Object (ADO) è una tecnologia simile, ma più recente, progettata per facilitare l'accesso remoto ai dati e le applicazioni client/server con OLE DB. Questo capitolo vi fornirà le informazioni di cui avrete bisogno per poter creare con successo applicazioni per i database con DAO e ADO.

Dai Data Access Object (DAO) agli ActiveX Data Object (ADO)

In Visual Basic sono disponibili tre interfacce di accesso ai dati:

- ActiveX Data Object (ADO)
- Remote Data Object (RDO)
- Data Access Object (DAO)

Una interfaccia di accesso ai dati è un modello a oggetti che è un livello di astrazione che gestisce i vari aspetti dell'accesso ai dati. Usando Visual Basic potete controllare da programma la connessione, la costruzione delle istruzioni e la restituzione dei dati per l'uso in qualsiasi applicazione mediante queste tre tecnologie.

Con la versione 6 di VB, la tecnologia ADO è matura e utilizzabile. È raccomandato che i nuovi progetti usino ADO invece che le altre due tecnologie più vecchie (RDO e DAO).

L'interfaccia che userete con ADO è simile all'interfaccia DAO, che tratterò in questo capitolo. È una logica progressione andare da DAO a ADO. RDO non è trattato in questo libro.

ODBC e OLE DB

Perché ADO? La risposta a questa domanda si trova nella transizione di Microsoft da ODBC a OLE DB, perché ADO è intesa come interfaccia facile da usare con OLE DB. ODBC, Open DataBase Connectivity, è una interfaccia standard tra un database ed una applicazione client che accede al database. Se la vostra connessione al database è fatta via ODBC, potete essere sicuri di avere un accesso SQL standardizzato.
Con OLE DB, Microsoft è andata oltre ODBC. OLE DB è pensato come sostituto di ODBC e per fornire accessi ad alte prestazioni a ogni tipo di sorgente di dati, ivi compresi i database relazionali e non, la posta elettronica e il file System, il testo e la grafica, oggetti personalizzati e altro ancora. OLE DB è progettato per rendere generiche tutte le sorgenti dei dati in modo che tutte le sorgenti di dati relazionali a cui si accede attraverso ODBC siano generiche. Mentre all'inizio OLE DB fu usato per accedere a database remoti su un Web server, con VB 6 può anche essere usato per applicazioni client/server generalizzate.
OLE DB non è progettato in modo che vi si possa accedere direttamente da Visual Basic a causa delle sue complesse interfacce: si passa invece attraverso ActiveX Data Oject (ADO), che incapsula ed espone virtualmente tutte le funzionalità di OLE DB.

Uso di DAO per lavorare con i database

DAO è una tecnologia ancora largamente usata e simile ad ADO. Questa sezione fornisce informazioni su come lavorare con DAO.

Ambienti dei database

DAO supporta due ambienti di database, chiamati anche *spazi di lavoro.* Lo spazio di lavoro Microsoft Jet permette di accedere ai dati in database di Microsoft Jet, in database ODBC connessi Microsoft Jet, e ad altre sorgenti di dati ISAM installabili in altri formati, come Paradox o Lotus 1-2-3.
Lo spazio di lavoro ODBCDirect permette di accedere ai server di database attraverso ODBC, senza dover caricare il motore per i database di Microsoft Jet. Usate lo spazio di lavoro Microsoft Jet quando aprite un database Microsoft Jet (file .mdb) o altri database ISAM del desktop, oppure quando volete usare le caratteristiche di Microsoft Jet, come l'abilità di raggnippare i dati da diversi formati di database.

Lo spazio di lavoro ODBCDirect fornisce una alternativa quando dovete eseguire solo delle query o delle procedure memorizzate su server back-end, come Microsoft SQL Server, oppure quando la vostra applicazione client deve usare le specifiche capacità di ODBC, come gli aggiornamenti batch o l'esecuzione asincrona delle query.

Oggetti DAO

Gli oggetti e le collezioni DAO permettono di creare e di manipolare i componenti all'interno di un sistema di database. Le proprietà degli oggetti e delle collezioni descrivono le caratteristiche dei componenti dei database. I metodi sono usati per manipolare i componenti. Gli oggetti e le collezioni DAO formano un modello gerarchico della struttura del vostro database, che potete usare per controllare la struttura.
Sono disponibili diciassette tipi di oggetti DAO, ognuno (ad eccezione di `DBEngine`) appartenente ad una collezione. La Tabella 32.1 elenca collezioni e oggetti, e descrive questi ultimi.

Tabella 32.1 *Oggetti DAO.*

Collezione	Oggetto	Descrizione
Connections	Connection	Informazioni su una connessione ad una sorgente dati ODBC (solo per gli spazi di lavoro ODBCDirect)
Containers	Container	Spazio per la memorizzazione di informazioni su un tipo di oggetto predefinito (solo per gli spazi di lavoro Microsoft Jet)
Databases	Database	Un database aperto
N/D	DBEngine	Il motore per i database Microsoft Jet
Documents	Document	Informazioni su un oggetto predefinito salvato (solo per gli spazi di lavoro Microsoft Jet)
Errors	Error	Informazioni su ogni errore associato con l'oggetto corrente
Fields	Field	Una colonna che fa parte di una tabella, di una query, di un indice, di una relazione, o di un recordset (l'analogo di una tabella per il motore Jet)
Groups	Group	Un gruppo di account utenti (solo per gli spazi di lavoro Microsoft Jet)
Indexes	Index	Ordinamento predefinito e unicità di valori in una tabella (solo per gli spazi di lavoro Microsoft Jet)
Parameters	Parameter	Un parametro per una query con parametri
Properties	Property	Una proprietà integrata o definita dall'utente
QueryDefs	QueryDef	Una definizione di query salvata
Recordsets	Recordset	I record in una tabella o query base
Relations	Relation	Una relazione tra i campi nella tabelle e le query (solo per gli spazi di lavoro Microsoft Jet)
TableDefs	TableDef	Una definizione di tabella salvata (solo per gli spazi di lavoro Microsoft Jet)
Users	User	Un account utente (solo per gli spazi di lavoro Microsoft Jet)
Workspaces	Workspace	Una sessione del motore per i database Microsoft Jet

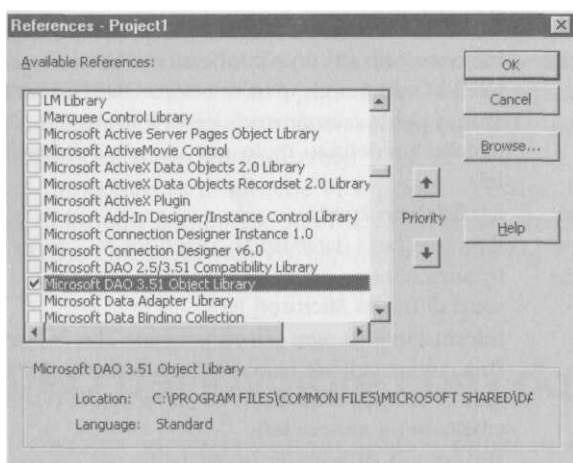
Dovete sapere che le collezioni delle classi DAO, come molte altre collezioni integrate in VB ma a differenza aelle collezioni da voi definite, sono a base zero: il primo elemento di una collezione DAO, cioè, è numerato con lo zero. Per maggiori informazioni sulle collezioni in VB, vedere il Capitolo 14.

Uso di DAO

Come potete vedere dalla Tabella 32.1, il modello degli oggetti DAO è esteso e potente. Molto comunemente, viene usato con database esistenti per eseguire query, aggiornare dei record e per la manutenzione del database.

Usando solo alcune proprietà e metodi dei più importanti oggetti e collezioni DAO (la dimostrazione salvata sul CD-ROM come Dao.Vdp utilizza solo DatabaseRecordset*), è possibile svolgere semplicemente molti compiti comuni per i database.*

Prima che iniziate ad usare gli oggetti DAO nei vostri progetti, dovrete aggiungere un riferimento alla DAO Object Library usando la finestra di dialogo *References,* come mostrato in Figura 32.1.

Figura 32.1
Prima di usare i membri della gerarchia DAO, dovete aggiungere un riferimento alla DAO Object Library al vostro progetto.

Connessione a un database

Per connettersi a un database mediante un oggetto DAO Database, bisogna prima dichiarare una variabile oggetto Database:

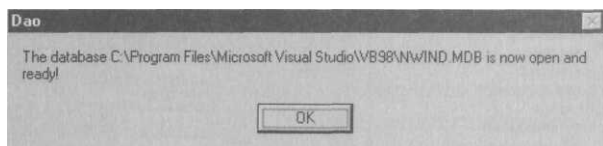
```
Dim db As Database
```

È importante considerare l'ambito di validità (scope) della variabile oggetto Database. *(Vedere il Capitolo 14 per informazioni generali sugli oggetti e sugli ambiti di validità in VB6.) Se avete una applicazione composta da un form, ha senso dichiarare la variabile nella sezione* General *del modulo, in modo che sia disponibile in ogni punto dell'applicazione.*

Se più form in una applicazione devono accedere allo stesso database, può aver senso usare un modulo di classe per gestire la connessione al database, istanziare un oggetto basato sulla classe quando l'applicazione parte e distruggere l'oggetto quando l'applicazione termina.
Per aprire un database, usate il metodo OpenDatabase dell'oggetto Workspace per assegnare un database alla variabile oggetto Database:

```
Private Sub  cmdOpen_Click()
   Set db = OpenDatabase("Nwind.mdb")
   MsgBox "The database " & db.Name & " is now open and ready!"
End Sub
```

Eseguendo questa porzione di codice con il database Nwind che viene distribuito con VB6, otterrete i risultati mostrati in Figura 32.2

Figura 32.2 *Per aprire un database, si usa il metodo OpenDatabase.*

Non è una buona idea aprire e chiudere una connessione a un database molte volte all'interno di una applicazione. Ogni volta che aprite un database, la vostra applicazione provocherà un aumento nell'utilizzo del sistema.

La sintassi del metodo OpenDatabase è:

```
OpenDatabase (dbname, [options], [readonly], [connect])
```

I parametri opzionali sono indicati con le parentesi quadre. La Tabella 32.2 descrive lo scopo dei parametri opzionali del metodo OpenDatabase.

Tabella 32.2 *Parametri opzionali del metodo OpenDatabase.*

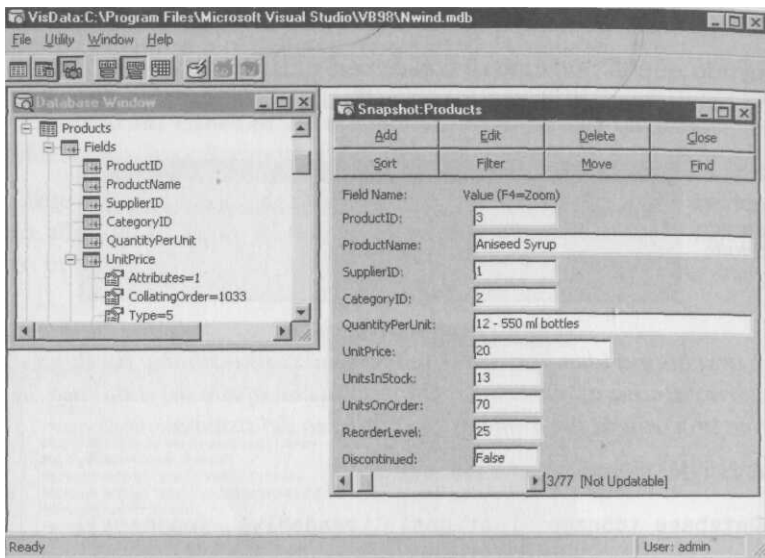
Parametro	Scopo
options	Usando lo spazio di lavoro Jet, se options viene valutata a True il database viene aperto in modalità esclusiva, nel senso che nessun altro può aprirlo mentre voi lo avete aperto. Se viene valutato a False, è aperto in modalità non esclusiva.
readonly	La connessione non può apportare modifiche al database se questo argomento è impostato a True.
connect	Usato con ODBCDirect per passare stringhe per la connessione ODBC.

Aggiornamento di una tabella

Per eseguire una istruzione SQL su un database aperto, usate il metodo Execute dell'oggetto Database. Ecco come potete raddoppiare UnitPrice per ogni record nella tabella Product di Nwind:

```
Private Sub cmdUpdate_Click()
  db.Execute "UPDATE Products " & _
     "SET UnitPrice = [UnitPrice]*2"
End Sub
```

Potete usare Visual Data Manager di VB, che è il primo elemento nel menu *Add-Ins* prima e dopo per verificare che i prezzi unitari siano stati effettivamente raddoppiati dopo aver eseguito questo metodo Execute. La Figura 32.3 mostra un record nella tabella Product in Nwind in Visual Data Manager.

Figura 32.3 *Potete usare Visual Data Manager per verificare che le modifiche siano state fatte nei record in una tabella.*

Creazione ed eliminazione di una tabella

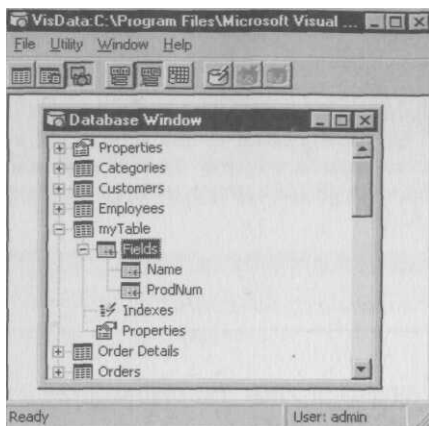
Il metodo Execute può essere usato anche per eseguire delle istruzioni Data Definition Language (DDL) su un database. (Vedere il Capitolo 31 per una discussione di DDL e degli altri sottolinguaggi SQL.)

Creazione di una tabella

Per esempio, il metodo Execute può essere usato per creare una tabella. Il nome della tabella è preso dal campo di testo di input txtTable, che è impostato per default a myTable:

```
Private Sub cmdAdd_Click()
   db.Execute "CREATE TABLE " & txtTable.Text & _
     "([Name] TEXT(50), [ProdNum] LONG)"
End Sub
```

Anche in questo caso potete usare Visual Data Manager per verificare che la tabella sia stata aggiunta, come mostrato nella Figura 32.4.

Figura 32.4 *Potete usare il metodo Execute con una strìnga DDL per creare una tabella.*

Mentre i campi, o le colonne, della tabella in questa istruzione DDL sono definiti rigidamente (Name *e* ProdNum) *niente vi impedisce di costruire una stringa che costruisca dinamicamente i nomi e i tipi dei campi ricevendoli dall'utente.*

Eliminazione di una tabella

Potete anche eliminare una tabella, usando una stringa DDL come argomento per il metodo Execute:

```
Private Sub cmdDelete_Click()
   db.Execute "DROP TABLE " & txtTable.Text
End Sub
```

Uso di un oggetto Recordset

In DAO, l'oggetto Recordset è usato per manipolare i record. Essenzialmente, un Recordset è come una tabella, ma nello spazio di lavoro Jet può anche essere costituito da record presi da più tabelle, oppure può essere una vista. Come con l'oggetto Database, il primo passo consiste nel dichiarare il Recordset.

```
Dim rs As Recordset
```

Poi si usa il metodo OpenRecordset dello spazio di lavoro per assegnare alla variabile una tabella, una vista, oppure una definizione di una query memorizzata, per esempio:

```
Set rs = db.OpenRecordset("Products")
```

Mentre il parametro del metodo OpenRecordset può essere semplicemente una sorgente di dati, può anche essere una istruzione SQL. Il Listato 32.1 mostra come potete aggiungere i nomi dalla tabella Employees di Nwind a una casella di riepilogo. Il codice nel listato costruisce una query basata su quanto immette l'utente in una casella di testo (il valore di default è "Davis"). Tutti i nomi che in ordine alfabetico vengono dopo il valore immesso sono aggiunti al Recordset. La collezione Fields degli oggetti Field è usata per aggiungere alla casella di riepilogo i nomi selezionati.

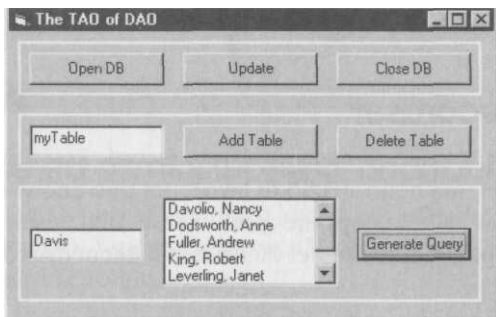
La punteggiatura conta!

Forse avete avuto un maestro d'italiano che vi diceva qualcosa del tipo "l'ortografia e la punteggiatura sono importanti!". La punteggiatura in una stringa SQL che è argomento di OpenRecordset è molto importante, in particolare se state usando l'input di un utente. A questo proposito, guardate gli apici singoli e doppi che racchiudono il campo txtName .Text nel Listato 32.1.

Listato 32.1 *Usare una query SQL con il metodo OpenRecordset.*

```
Private Sub cmdQuery_Click()
   Set rs = db.OpenRecordset("SELECT * FROM Employees " & _
      "WHERE [LastName] > "' & txtName.Text &" ' " &_
      "ORDER BY [LastName]")
   Do Until rs.EOF
      List1 .AddItem rs.Fields( "LastName" ) &_
         ", " & rs.Fields("FirstName")
      rs.MoveNext
   Loop
End Sub
```

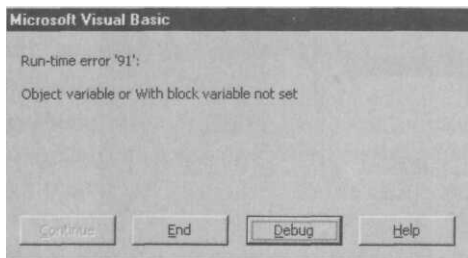
I possibili risultati dell'esecuzione di questo codice sono mostrati in Figura 32.5.

Figura 32.5 *Potete usare il metodo OpenRecordsete la collezione Fields ber aggiungere i record ad una casella di riepilogo.*

Chiusura della connessione

Come ogni altra variabile oggetto, l'oggetto Database, che rappresenta la connessione al database, viene distrutto quando l'ultimo riferimento all'oggetto esce dall'ambito di validità: si parla anche di *terminazione implicita.* (Vedere il Capitolo 14 per una descrizione più dettagliata di questi concetti.)
Per esempio, nel semplice esempio con un form in Dao.Vbp, l'oggetto Database è terminato implicitamente quando il form è chiuso. Potete comunque gestire la connessione al database e terminarla esplicitamente in qualche punto. Per terminare esplicitamente una connessione a un database, impostate a Nothing la variabile oggetto che contiene la connessione:

```
Private Sub cmdClose_Click()
   Set db = Nothing
End Sub
```

Una volta che una variabile oggetto Database è stata distrutta, se un utente tenta di usare i metodi dell'oggetto Database, verrà generato il famigerato errore 91, "Object variable or With block variable not set", come mostrato nella Figura 32.6. Per informazioni su come gestire questo errore, vedere il Capitolo 15.

Figura 32.6 *Se tentate di usare i metodi di un oggetto Database che è stato deistanziato, otterrete un errore sulla variabile Object.*

Che cosa sono gli ActiveX Data Object

ADO, ActiveX Data Object, è inteso come sostituto per DAO (e RDO).

Con VB6, ADO viene distribuito per la prima volta come pane integrante dell'ambiente Visual Basic.

Essenzialmente, ADO aggiunge un nuovo livello, OLE DB, tra una sorgente di dati ODBC e l'applicazione client Visual Basic. OLE DB serve per fornire un accesso astratto ad alte prestazione a qualsiasi sorgente di dati, compresi i database relazionali e non.

Secondo Microsoft, "ADO viene implementato con un piccolo carico, con un traffico di rete minimo per scenari Internet, e con un numero minimo di livelli tra il front-end e la sorgente dei dati, tutto per fornire una interfaccia leggera dalle alte prestazioni. ADO è facile da usare perché viene chiamato usando una metafora familiare, l'interfaccia OLE Automation, disponibile in ogni strumento e linguaggio attualmente presente sul mercato. E poiché ADO è stato progettato per combinare le caratteristiche migliori di, e per eventualmente rimpiazzare RDO e DAO, usa una convenzione simile con una semantica semplificata che è facile da imparare per i moderni sviluppatori".

La struttura a oggetti ADO è molto simile alla struttura a oggetti DAO (che è stata spiegata in precedenza in questo capitolo), ma è meno gerarchica e più lineare nella sua natura. Tutti gli oggetti ADO, ad eccezione degli oggetti Error e Field, possono essere creati indipendentemente da ogni altro oggetto ADO. Questo permette di creare oggetti che possono essere riutilizzati in contesti diversi. Per esempio, potete creare un oggetto Command, associarlo ad una connessione ed eseguirlo, poi associarlo con una diversa connessione ed eseguirlo li. Per poter usare ADO in una applicazione client Visual Basic, dovete aggiungere ADO Data Control attra-

verso la finestra di dialogo *Components* (vedere Figura 32.7) oppure ADO Library dalla finestra di dialogo *References* (vedere Figura 32.8).

Figura 32.7 *La finestra di dialogo Components è usata per aggiungere ADO Data Control al vostro Toolbox.*

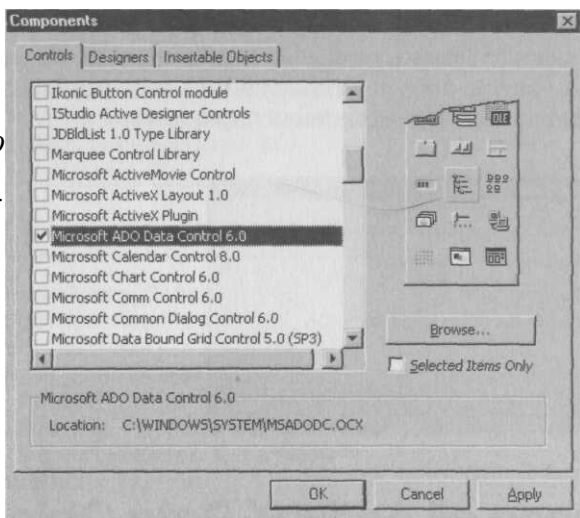
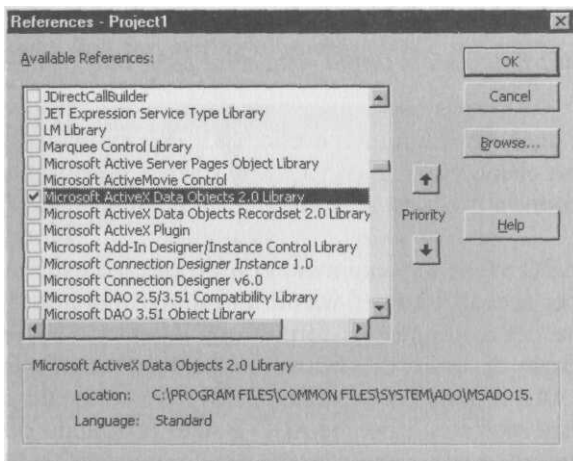



Figura 32.8 *Potete usare gli oggetti ADO nel codice se prima usate la finestra di dialogo References per includere la ADO Library.*

I sei oggetti ADO principali sono il seguenti:

- **Oggetto Connection.** Questo oggetto rappresenta una connessione a una sorgente di dati e permette di eseguire i comandi. Per eseguire qualsiasi tipo di comandi, usate il metodo `Execute` dell'oggetto `Connection`. Se il comando restituisce delle righe, viene creato e restituito un Recordset di default.
- **Oggetto Gommand.** Questo oggetto è un comando (come una query o una istruzione) che può essere elaborato dalla sorgente dei dati. I comandi possono restituire delle righe o meno e, a seconda del database, possono

anche gestire dei parametri. L'oggetto Command è opzionale nel modello ADO poiché alcune sorgenti dei dati non possono fornire l'estensione per l'esecuzione dei comandi, ma l'oggetto è supportato se chi fornisce i dati supporta i comandi. I comandi possono essere semplici istruzioni SQL (o di altri linguaggi che il fornitore dei dati riconosce) oppure delle chiamate a procedure memorizzate nel database. I comandi possono essere eseguiti mediante il metodo Execute di Command.

Gli oggetti Command includono una collezione di oggetti Parameter, che sono descritti di seguito. Se la sorgente può supportare i comandi con parametri, la collezione Parameters conterrà un oggetto Parameter per ogni parametro del comando.

- **Oggetto Parameter.** Ognuno è un parametro di un Command. Come ho spiegato nella descrizione dell'oggetto Command, potete creare esplicitamente gli oggetti Parameter e aggiungerli alla collezione Parameters per evitare il compito spesso inutile e costoso di andare nel catalogo di sistema del database per mettere automaticamente le informazioni sui parametri.
- **Oggetto Recordset** Questo è senz'altro l'oggetto ADO più complesso. Assomiglia a quello presente in DAO, ma sono stati apportati dei miglioramenti, come la rimozione degli elementi non necessari, l'aggiunta di argomenti opzionali che riducono il numero di righe di codice per le situazioni più comuni, e la modifica dei default che non hanno senso nelle moderne tecnologie.
- **Oggetto Field.** Questo è una colonna in un Recordset che potete usare per ottenere valori, per modificare i valori e per recuperare informazioni sulle colonne. Questo oggetto è quasi identico all'oggetto Field in DAO.
- **Oggetto Error.** Questo oggetto contiene un errore restituito da una sorgente dei dati e spesso non viene usato perché è necessario solo quando una sorgente dei dati può restituire più errori per una singola chiamata a un metodo. Se una sorgente non restituisce più errori per una singola chiamata a funzione, questa può fornire l'errore attraverso i normali meccanismi ActiveX che tutti i server ActiveX usano quando vengono chiamati da linguaggi come Visual Basic.

Il controllo Data ADO

Molto simile nel modo di funzionare al controllo Data descritto nel Capitolo 31, il controllo Data ADO vi permette di collegare i controlli, come il controllo DataGrid ActiveX, a una sorgente di dati OLE DB senza scrivere neanche una riga di codice.

Il *controllo* Data *ADO, così come viene fornito, comprende i pulsanti avanti, indietro, vai all'inizio e vai alla fine, che potete usare per navigare attraverso una sorgente di dati.*

È possibile creare una applicazione di database con una minima quantità di codice impostando alcune proprietà durante la fase di progettazione. Per cominciare, aggiungete un controllo Data ADO al form. Usando la pagina *General* della finestra di dialogo delle proprietà personalizzate del controllo, come mostrato in Figura 32.9, impostate la sorgente dei dati.

Nell'esempio salvato sul CD-ROM come Ado. Vbp, ho usato il database blbliografico di Access distribuito con VB6.

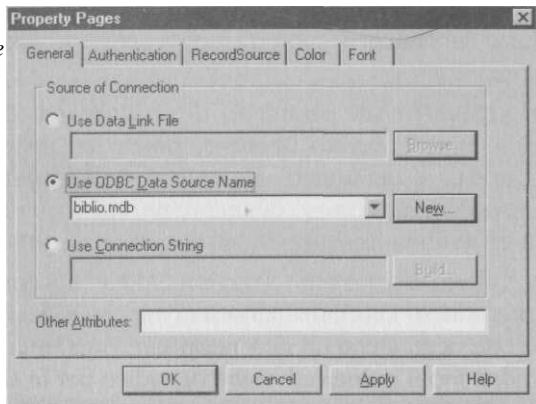
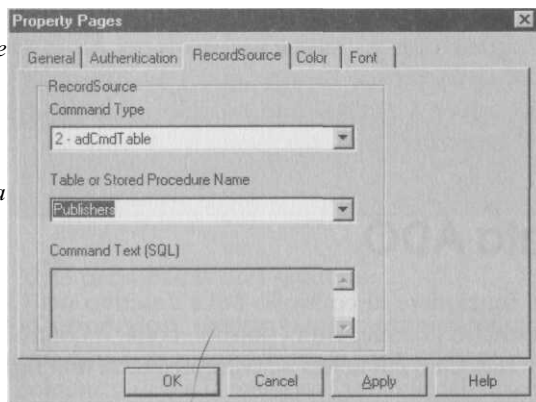
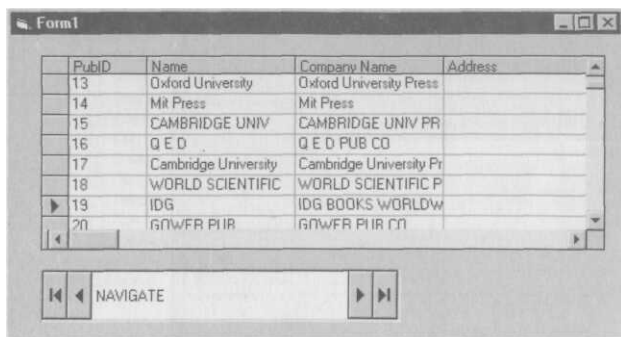
Figura 32.9 *Poteteimpostare la sorgente dei dati nella scheda General della finestra di dialogo Property Pages del controllo Data ADO.*

Di seguito, potete usare la scheda *RecordSource* della *Property Pages* del controllo per impostare *Command Type* e *Table or Stored Procedure Name.* Nel mio esempio, mostrato in Figura 32.10, l'ho impostato alla tabella Publishers nella sorgente dei dati blblio. Notate che se *Command Type* è stato impostato a 1-adCmdText, dovete immettere direttamente una stringa SQL per l'esecuzione.

Figura 32.10 *Poteteimmettere una tabella, una procedura memorizzata, oppure una istruzione SQL usandolascheda RecordSource.*

Infine, aggiungete un controllo DataGrid (elencato come *Microsoft DataGrìd Control 6.0* nella finestra di dialogo *Componente).* Impostate la proprietà DataSource del DataGrid al controllo Data ADO, Adodc1. Quando eseguite il progetto, il controllo Data ADO riempirà il DataGrid. Potete usare le frecce del controllo ADO per

navigare nei record della griglia, come mostrato in Figura 32.11. Il record corrente nella griglia è mostrato con un indicatore nella colonna più a sinistra della griglia.

Figura 32.11 *potete collegare un controllo DataGrid al controllo Data ADO perpermettere di navigare facilmente all'interno di una griglia.*

Data Environment

Data Environment è uno strumento di progettazione che permette di creare e di manipolare visualmente le connessioni e i comandi ADO. Il Data Environment designer è usato per creare un oggetto DataEnvironment. L'oggetto DataEnvironment può comprendere oggetti ADO Connection e Command e gruppi gerarchici di oggetti Command.

Per aggiungere un Data Environment designer al vostro progetto, selezionare More ActiveX Designers | Data Environment *dal menu* Project *di VB. Per accedere ai membri dell'oggetto* DataEnvironment *da un progetto VB, assicuratevi che Data Environment sia abilitato nella finestra di dialogo* References *delprogetto.*

Quando aggiungete un Data Environment designer al vostro progetto, verrà aperta automaticamente una finestra di dialogo *Connection Properties,* come mostrato nella Figura 32.12. Questa è la stessa finestra di dialogo *Connections* che fornisce il controllo Data ADO (vedere la sezione precedente). Sono supportate sia le sorgenti di dati OLE DB che ODBC.
Una volta che avete fornito la sorgente dei dati per la connessione, potete manipolare gli oggetti Command e Connection usando il Data Environment.

Data Environment supporta più oggetti Connection permettendovi di accedere a più sorgenti di dati all'interno di un singolo ambiente di dati. Per aggiungere una nuova Connection,*fate clic con ilpulsante sinistro del mouse sulla barra degli strumenti del Data Environment. Verrà apena una nuova connessione.*

Quando create un oggetto Command, dovrete specificare quale oggetto Connection usa, scegliendo un oggetto Connection dall'elenco nella finestra di dialogo *CommandProperties,* come mostrato nella Figura 32.13. Gli oggetti nel Data Environment possono essere organizzati per oggetti (vedere Figura 32.14) o per connessione (vedere Figura 32.15).

Figura 32.12

Una finestra di dialogo Connection Properties viene automaticamente aperta quando aggiungete un Data Environment designer al vostro progetto.

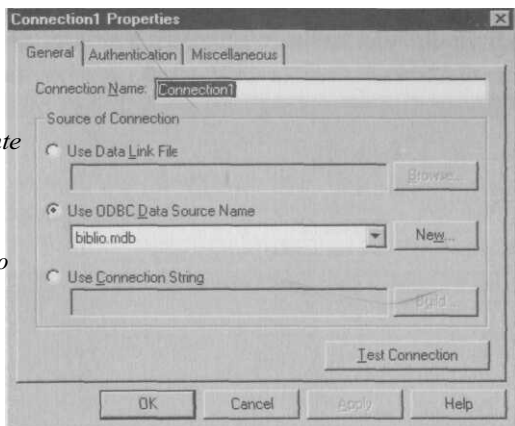



Figura 32.13

Ogni oggetto Command è collegato ad un oggetto Connection, come specificato nella finestra di dialogo Properties dell'oggetto Command.

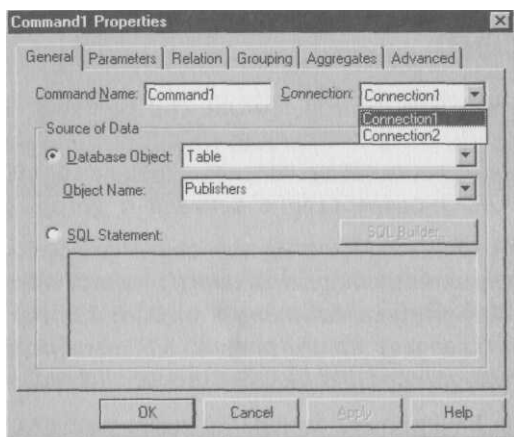



Figura 32.14

Se il Data Environment è organizzato per oggetti, allora le Connection e i Command sono elencati separatamente.

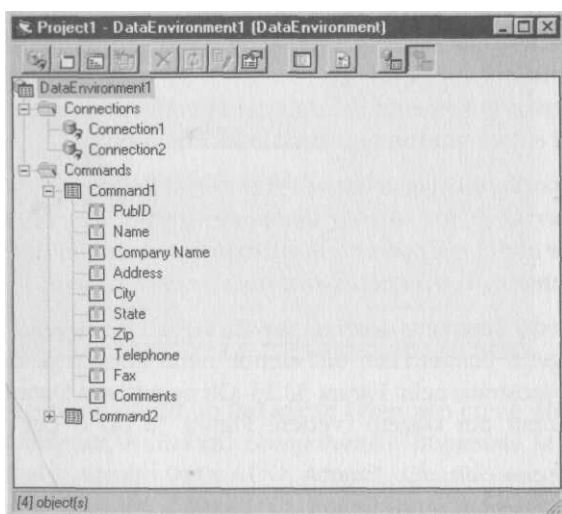
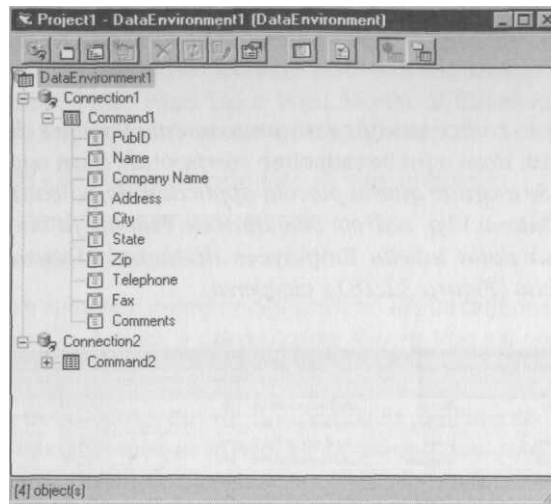
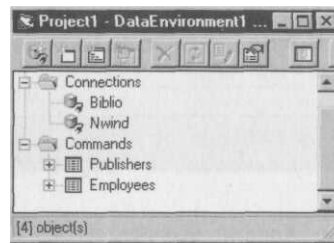



Figura 32.15
Quando il Data Environment è organizzato per connessione, allora i comandi sono mostrati sotto la connessione alla quale sono collegati.

Potete trascinare i campi e le tabelle dal vostro Data Environment designer in un form o nel Data Report ActiveX designer. I controlli collegati ai dati sono creati automaticamente nel form.

Una delle cose comode di Data Environment è che potete programmare usando un Data Environment che è collegato ai controllo in un form senza il riferimento a variabili. Per esempio, supponiamo che io abbia un Data Environment (mostrato in Figura 32.16) con due Connection, chiamate Blblio e Nwind, ognuna collegata ai corrispondenti database. Ho creato due oggetti Command nel Data Environment: Publishers, che è la tabella *Publishers* del database Blblio, e Employees, che è la tabella *Employees* del database Nwind.

Figura 32.16
Un Data Environment può contenere oggetti Command collegati a diversi database.

Potete aggiungere un controllo DataGrid, *Microsoft DataGrid Control 6.0* nella finestra di dialogo *Components,* e collegarlo al Data Environment impostando la proprietà DataSource del DataGrid all'oggetto DataEnvironment. Poi, potete aggiungere un pulsante di comando e il codice per passare tra le due tabelle, nei diversi database, durante l'esecuzione:

```
Private Sub Command1_Click()
   If DataGridl.DataMember = "Publishers" Then
      DataGridl.DataMember = "Employees"
```

```
    Else
        DataGridl.DataMember = "Publishers"
    End If
End Sub
```

Come potete vedere, questo codice modifica dinamicamente il valore della proprietà DataMember *del* DataGrid, *dove ogni* DataMember *corrisponde a un oggetto* Command *del Data Environment. Se eseguite questa piccola applicazione, salvata sul CD-ROM allegato al libro come Datenv.Vbp, vedrete che, quando l'utente fa clic sul pulsante* Change, *la griglia passa dalla tabella* Employees *di Nwind (Figura 32.17) alla tabella* Publishers *di Blblio (Figura 32.18) e viceversa.*

Figura 32.17

Potete usare la proprietà DataMember del controllo DataGrid per visualizzare i dati determinati dall'oggetto Command del Data Environment.

DataGrid bound to a DataEnvironment

EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate	Hir
1	Davolio	Nancy	Sales Representative	Ms.	12/8/48	5/1
2	Fuller	Andrew	Vice President, Sales	Dr.	2/19/52	8/1
3	Leverling	Janet	Sales Representative	Ms.	8/30/63	4/1
4	Peacock	Margaret	Sales Representative	Mrs.	9/19/37	5/3
5	Buchanan	Steven	Sales Manager	Mr.	3/4/55	10/
6	Suyama	Michael	Sales Representative	Mr.	7/2/63	10/
7	King	Robert	Sales Representative	Mr.	5/29/60	1/2
8	Callahan	Laura	Inside Sales Coordinator	Ms.	1/9/58	3/5
9	Dodsworth	Anne	Sales Representative	Ms.	1/27/66	11/

Change

Figura 32.18

Cambiando il DataMember in un secondo oggetto Command del Data Environment, potete visualizzare un altro insieme di dati da una sorgente di dati separata.

DataGrid bound to a DataEnvironment

PubID	Name	Company Name	Address	City	State
1	SAMS	SAMS	11711 N. College Ave.,	Carmel	IN
2	PRENTICE HALL	PRENTICE HALL	15 Columbus Cir.	New York	NY
3	M & T	M & T BOOKS			
4	MIT	MIT PR			
5	MACMILLAN COMPUTI	MACMILLAN COMPUTI	11 W. 42nd St., 3rd flr.	New York	NY
6	HIGHTEXT PUBNS	HIGHTEXT PUBNS			
7	SPRINGER VERLAG	SPRINGER VERLAG			
8	O'REILLY & ASSOC	O'REILLY & ASSOC	90 Sherman St.	Cambridge	MA
9	ADDISON-WESLEY	ADDISON-WESLEY PL	Rte 128	Reading	MA
10	JOHN WILEY & SONS	JOHN WILEY & SONS	605 Third Ave	New York	NY
11	SINGULAR	SINGULAR PUB GROL			
12	Duke Press	Duke Press			
13	Oxford University	Oxford University Press			

Change

Il controllo DataRepeater

Il controllo DataRepeater, *Microsoft DataRepeater Control 6.0* nella finestra di dialogo *Components, è* usato come involucro legato ai dati per un controllo ActiveX personalizzato. È usato generalmente con i controlli ActiveX che sono stati creati mediante controlli costituenti (vedere il Capitolo 26 per una spiegazione).

L'idea è che il controllo ActiveX personalizzato è progettato per mostrare un singolo record di un database. La proprietà RepeatedControlName del DataRepeater è impostata al controllo ActiveX, e il DataRepeater è collegato alla sorgente dei dati mediante un controllo Data ADO.

Quando l'applicazione viene eseguita, il DataRepeater visualizza più istanze del controllo ActiveX che contiene, ognuna nella propria riga, e ognuna collegata ad un diverso record del database. L'utente può scorrere diversi record usando le frecce ed i tasti Home, Fine, Page Up e Page Down. Il DataRepeater ha vari impieghi, quando viene impostato in questo modo, fra cui:

- Cataloghi che comprendono foto di ogni elemento
- Applicazioni finanziarie e bancarie
- Programmi di inventario

Ho salvato un semplice esempio dell'utilizzo del DataRepeater *sul CD-ROM. Ilprogetto del controllo ActiveX è salvato come Repeat. Vbp e ilprogramma che usa il controllo ActiveX con il DataRepeater è Testrep.Vbp.*

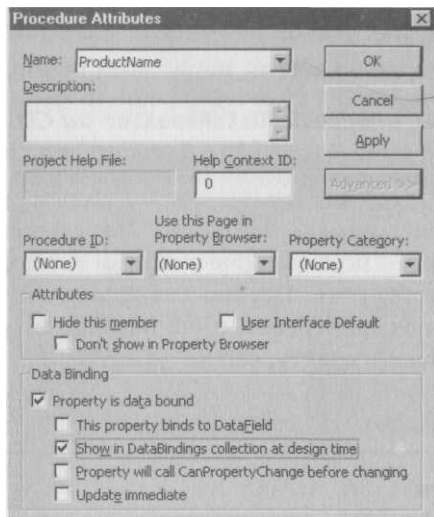
Repeat.Vbp è un progetto di un controllo dell'utente. Ho cambiato il nome dell'oggetto UserControl in myRep, ed ho messo due caselle di testo, txtProductName e txtUnitPrice, nello UserControl. Devono essere aggiunte al modulo alcune procedure Property Get e Property Let, come mostrato nel Listato 32.2. (Per una spiegazione delle procedure Property, vedere il Capitolo 14.)

Listato 32.2 *Aggiungere leprocedure delleproprietà al UserControl.*

```
Public Property Get ProductName() As String
   ProductName = txtProductName.Text
End Property

Public Property Let ProductName(ByVal newProductName As String)
   txtProductName.Text = newProductName
End Property

Public Property Get UnitPrice() As String
   UnitPrice = txtUnitPrice.Text
End Property

Public Property Let UnitPrice(ByVal newUnitPrice As String)
   txtUnitPrice.Text = newUnitPrice
End Property

Private Sub txtProductName_Change()
   PropertyChanged "ProductName"
End Sub

Private Sub txtUnitPrice_Change()
   PropertyChanged "UnitPrice"
End Sub
```

UnitPrice *è dichiarato nel codice mostrato nel Listato 32.2 come* String. *Probabilmente vi aspettavate che fosse dichiarato come* Currency. *Però il progetto usa l'oggetto* DataFormat *per formattare correttamente il campo. Il risultato di un oggetto* DataFormat *è sempre una stringa, quindi se la proprietà fosse stata dichiarata come* Currency *avrebbe modificato il formato.*

Poi bisogna rendere *data-bound* (legate ai dati), le proprietà UnitPrice e Product Name. Per fare questo, aprite la finestra di dialogo *Procedure Attributes* dal menu *Tools*. Fate clic sul pulsante *Advanced*. Come mostrato nella Figura 32.19, per entrambe le procedure selezionate *Property is data bound* e *Show in DataBindings collection at design lime.*

Figura 32.19 *La finestra di dialogo Procedure Atributesèusata per collegare una proprietà ai dati.*

Compilate il controllo, che viene così registrato nel vostro sistema, permettendogli di essere usato con il controllo DataRepeater. Aprite un nuovo progetto standard. Aggiungete un controllo Data ADO e un controllo DataRepeater al progetto.

Dovete disegnare il DataRepeater *con una larghezza sufficiente a contenere più istanze del controllo ActiveX personalizzato.*

Impostate la sorgente dei dati del controllo Data ADO al database Nwind, e Record-Source alla tabella *Products,* come descritto in precedenza in questo capitolo. Usando la finestra *Properties,* impostate la proprietà DataSource del DataRepeater al controllo Data ADO, chiamato per default Adodc1. Scorrete la proprietà DataSource del DataRepeater. Nella casella di riepilogo troverete tutti i controlli disponibili per ProgID. Selezionate il controllo Activex che è appena stato creato, Repeat.myRep.
Poi collegate le proprietà del controllo utente al controllo Data ADO. Aprite la finestra di dialogo delle proprietà *Custom* del controllo DataRepeater. Selezionate la scheda *RepeaterBindings.* Come mostrato nella Figura 32.20, assegnate ogni proprietà all'appropriato campo.

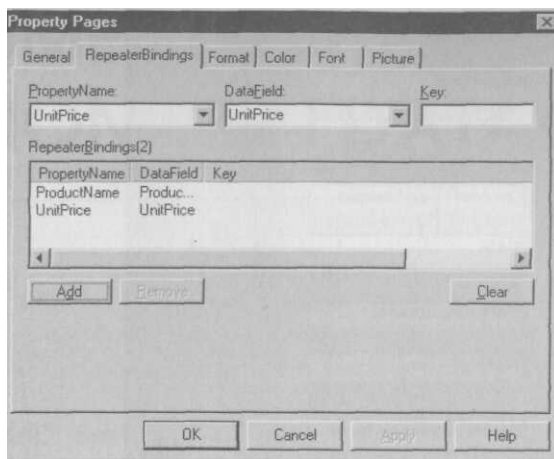
Figura 32.20 La scheda *RepeaterBindings della finestra di dialogo delle proprietà Custom del controllo DataRepeater è usata per collegare le proprietà del controllo utente ai campi.*

Selezionate la scheda *Format* della finestra di dialogo delle proprietà personalizzate per usare gli oggetti DataFormat per formattare correttamente il campo UnitPrice, come mostrato nella Figura 32.21. Se eseguite il progetto, sarete in grado di sfogliare i contenuti della tabella *Products,* come mostrato nella Figura 32.22.

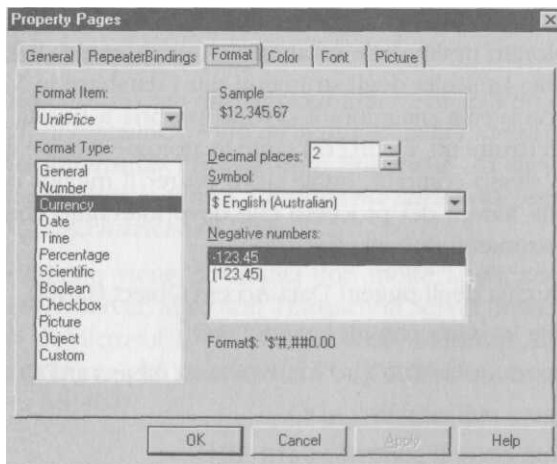
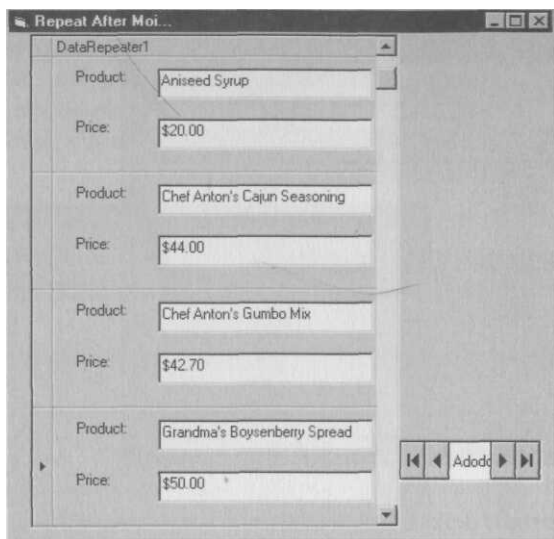
Figura 32.21 *Gli oggetti DataFormat sono usatiper collegare . le sorgenti dei dati con le formattazioni appropriate.*

Potete impostare le opzioni di formattazione mediante la scheda Format *della finestra di dialogo delle proprietà personalizzate di un controllo legato. Questa finestra di dialogo rappresenta la proprietà* DataFormat *del controllo legato, che imposta o restituisce l'oggetto* StdDataFormat. *Quando il controllo legato recupera un record dalla sorgente dei dati, viene applicata la formattazione secondo le proprietà dell'oggetto* StdDataFormat. *Poi viene generato un evento* Format *e il dato formattato è visualizzato dal controllo legato.*

Figura 32.22
Il *controllo DataRepeater è un utile strumento per permettere agli utenti di sfogliare i contenuti di una sorgente di dati.*

Riepilogo

Questo capitolo ha esplorato molte delle caratteristiche importanti dell'arsenale per i database di Visual Basic. La scelta degli strumenti per i database in Visual Basic è molto ampia. Infatti il problema maggiore è dovuto propria a questa ricchezza di strumenti. Con così tanti strumenti, e con così tante tecnologie incluse nel pacchetto di VB che eseguono lo stesso compito, quale devo usare? Il miglior consiglio è di pensare chiaramente alla natura del processo che deve interagire con la sorgente dei dati, e scegliere gli strumenti di conseguenza.

- Ho spiegato la gerarchia degli oggetti Data Access Object (DAO).
- Vi ho mostrato come lavorare con gli oggetti DAO.
- Ho descritto la transizione da DAO ad ActiveX Data Object (ADO).
- Ho spiegato gli oggetti nell'universo ADO.
- Avete imparato come usare il controllo Data ADO.
- Ho spiegato il Data Environment.
- Vi ho insegnato come usare il Data Environment per manipolare più sorgenti di dati.
- Ho mostrato una applicazione per sfogliare i dati usando un controllo ActiveX personalizzato ed il controllo DataRepeater.

STRUMENTI ENTERPRISE EDITION PER I DATABASE

- Microsoft SQL Server 7.0
- Microsoft Transaction Server 2.0
- Microsoft Visual Modeler

Nel Capitolo 31 e nel Capitolo 32 ho descritto alcuni degli strumenti per i database disponibili agli sviluppatori Visual Basic. Questo capitolo fornisce una introduzione alle importanti applicazioni e tecnologie per i database disponibili agli utenti della Enterprise Edition di Visual Basic 6. L'analisi completa di questi strumenti richiederebbe altri volumi supplementari, ma questo capitolo vi dovrebbe almeno fornire una idea di quello che sono.

Microsoft ha dato un prezzo alle versioni Enterprise di VB6 e di Visual Studio 6 in modo che molti sviluppatori pronti all'acquisto della Enterprise Edition di VB6, probabilmente acquisteranno l'intera suite Enterprise. In alcuni casi, le tecnologie descritte in questo capitolo fanno, strettamente parlando, parte della Enterprise Edition di Visual Studio anziché di VB6.

La Enterprise Edition viene distribuita con molte applicazioni di Microsoft Back Office, fra cui SQL Server, Microsoft Transaction Server, Internet Information Server (il server Web di Microsoft), Visual SourceSafe (descritto nel Capitolo 12) e SNA Server (un programma che permette ai PC di comunicare con mainframe remoti e con i computer AS/400).

Microsoft SQL Server 7.0

SQL Server è un server di database aziendali progettato per essere eseguito in Windows NT. SQL Server viene eseguito come *servizio* di Windows NT. Questo significa che si usa *MSSQL Service* nell'applet *NT Services* nella finestra *Control Panel* per avviare e per fermare il SQL Service. In alternativa, si può usare l'applicazione SQL Service Manager, che si trova nel gruppo di programmi SQL Server, per eseguire le stesse operazioni.

Una versione per lo sviluppo di SQL Server 7.0 viene fornita con l'Enterprise Edition di VB6. È una versione completa, ma è limitata a 15 connessioni di client concorrenti. Questo solitamente è sufficiente per scopi di sviluppo, ma per distribuire una applicazione basata su SQL Server dovrete acquistare le licenze per i client aggiuntivi.

Ogni server di database aziendale, e SQL Server non fa eccezione, è sia uno stile di vita sia uno strumento che richiede molte risorse. "Stile di vita" significa che questa è una applicazione come poche altre che avete potuto incontrare, perché si prende carico di una vasta gamma di operazioni complesse che comprendono i compiti normalmente gestiti da un sistema operativo. Potrete avere dei problemi iniziali di apprendimento se non avete mai lavorato prima con i server dei database d'impresa.

È difficile quantificare, ma ricordatevi che le risorse sono un problema. Per eseguire tranquillamente SQL Server, avrete bisogno di una CPU veloce o di più CPU, una notevole quantità di spazio su disco, e probabilmente un minimo di 128 MB di RAM.

Potete interagire con SQL Server usando uno strumento GUI, Enterprise Manager, oppure immettendo istruzioni SQL. In entrambi i casi, i passi generali per poter far funzione SQL Server sono:

- Creare una unità disco, che è una astrazione, chiamata anche disco *logico,* che può coprire più unità disco fisiche
- Creare un database in una unità disco
- Creare le tabelle nel database
- Creare le viste e le procedure archiviate che controllano come i dati sono recuperati dal database
- Impostare gli utenti e i gruppi di utenti per scopi di sicurezza

Microsoft Transaction Server 2.0

Microsoft Transaction Server (MTS) è un sistema di elaborazione delle transazioni basato su componenti pensato per lavorare con Internet Information Server. MTS è usato per creare, distribuire e amministrare applicazioni Internet e intranet client/server a tre livelli usando componenti ActiveX. MTS offre l'accesso ai più diffusi server di database, SQL Server compreso.

L'architettura client/server a tre livelli è fondamentale in applicazioni Web per l'azienda perché le informazioni riservate nei database risiedono dietro i firewall. È compito del livello intermedio distribuire le informazioni all'esterno del firewall in modo che gli utenti vi possano accedere dai loro browser.

MTS offre componenti per la gestione (come il supporto automatico per le transazioni), caratteristiche di sicurezza, accesso a database noti, prodotti per accodare i

messaggi e applicazioni basate su mainframe, e caratteristiche per migliorare le prestazioni come il "pooling" di connessioni a database.
potete richiamare i componenti basati su MTS da script Internet Information Server ASP (Active Server Pages). MTS Explorer, una interfaccia grafica, può essere usata per distribuire e per gestire i componenti. Gli strumenti di sviluppo all'interno di MTS comprendono applicazioni d'esempio, un'ampia interfaccia di programmazione per le applicazioni (API) e la capacità di creare dei *dispensatori di risorse.* Un dispensatore di risorse è un servizio che gestisce uno stato condiviso temporaneo per componenti di più applicazioni.

Visual Modeler

Visual Modeler è uno strumento grafico usato per creare, o per determinare, le relazioni degli oggetti, dei componenti e dei dati nelle applicazioni Visual Basic (o di Visual C++).

Visual Modeler è un sottoinsieme dello strumento di modellazione Rational Rose. Entrambi i prodotti sono stati creati da Rational Software Corporation. Potete trovare più informazioni su Rational Rose al sito Web Rational, http://www.rational.com.

Le caratteristiche più importanti di Visual Modeler sono:

- La capacità di progettare un sistema costruendo un modello mediante diagrammi per le classi. La notazione a diagrammi che usa Visual Modeler è basata sui costrutti di modellazione astratta definiti dallo Unified Modeling Language (UML).
- Generazione di codice automatico. Visual Modeler genera automaticamente codice Visual Basic o Visual C++ basandosi sui diagrammi che avete creato.
- Processo a due vie, chiamato anche *reverse engineering.* La capacità di aggiornare (o di creare) automaticamente un modello (ed i diagrammi) basandosi su modifiche al codice del progetto Visual Basic.

Il *termine* round-trip engineering *è usato per indicare la combinazione della modellazione, della generazione del codice, della codifica e del* reverse engineering.

Visual Modeler può essere avviato come applicazione a sé stante dal menu *Tools* di Microsoft Visual Studio 6.0 Enterprise. Potete anche abilitarlo come add-in di VB6 attraverso l'Add-In Manager.

Per usare Visual Modeler dall'interno di Visual Basic, dovete caricare il Visual Modeler Add-In e il Visual Modeler Menus Add-In.

Se aprite un progetto di Visual Basic e selezionate *Reverse Engineering Wizard* dalla voce *Visual Modeler* dal menu *Add-Ins,* sarete in grado di selezionare gli elementi del progetto che devono essere inclusi nel *reverse engineering,* come mostrato nella Figura 33.1. Poi, assegnate ogni elemento ad un pacchetto logico nel modello, come mostrato nella Figura 33.2

Figura 33.1 *Reverse Engineering Wizard vipermette di selezionare gli elementi del progetto per l'inclusione nel modello.*

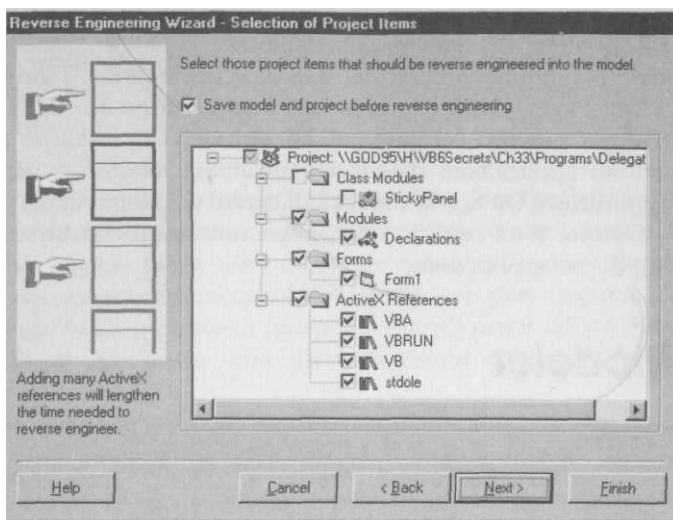
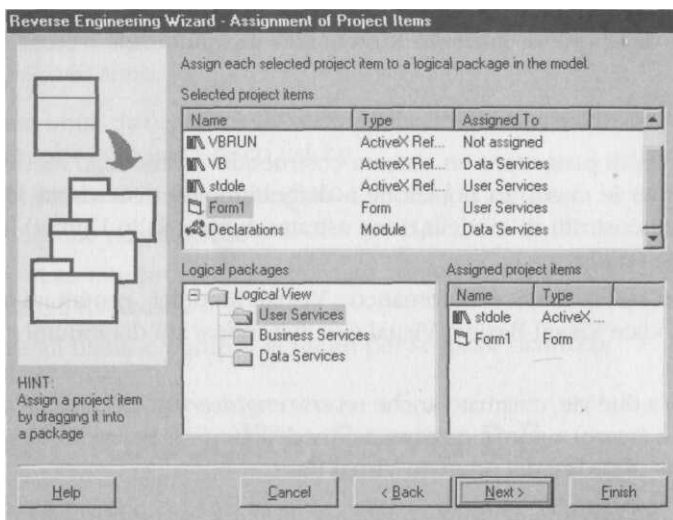



Figura 33.2 *Ogni elemento nel modello è assegnato ad un pacchetto logico.*

Visual Modeler è stato pensato per essere usato con progetti realizzati con un modello client/server a tre livelli. Data Services è il livello che interagisce con le sorgenti dei database, che si trovano presumibilmente dietro un firewall per sicurezza. I servizi Business sono componenti che forniscono dati con cui l'applicazione può interagire e codificano le "regole aziendali". User Services fornisce l'interfaccia utente e interagisce con i componenti del servizio Business.

Prima che il Wizard crei o aggiorni effettivamente un modello di progetto, visualizza una schermata con gli elementi del progetto e il numero di classi e membri in ognuno di essi (vedere Figura 33.3). Viene fornito anche un tempo stimato; in alcuni casi questo può essere abbastanza grande.

Figura 33.3

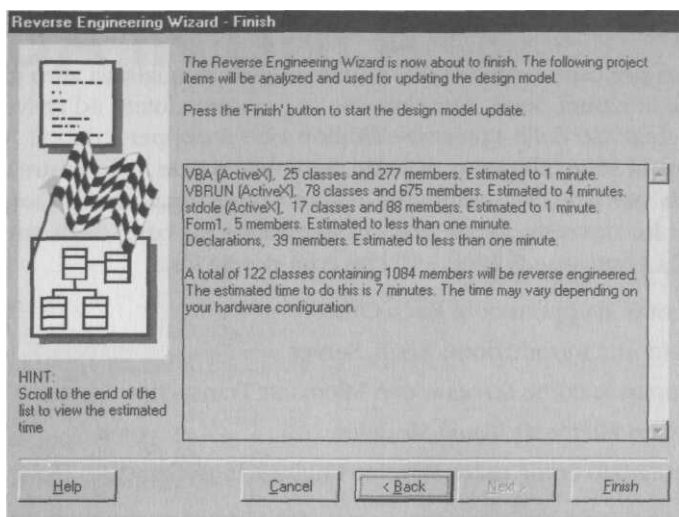
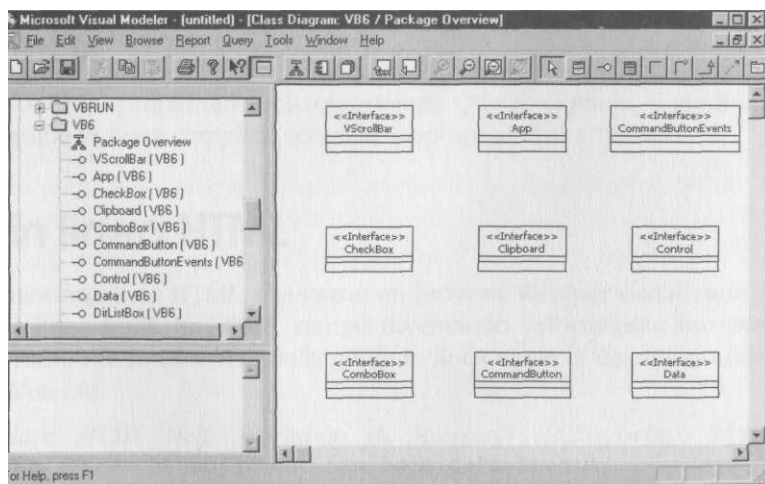
Reverse Engineering Wizard fornisce il tempo stimato prima di creare un modello.

Una volta che il modello visuale dell'applicazione è stato creato, può essere visualizzato in molti modi diversi. *Logicai View* visualizza un modello del servizio a tre livelli. *Package Overview,* mostrato in Figura 33.4, mostra i membri di un pacchetto. *Deployment View* mostra i nodi di implementazione dell'applicazione.

Figura 33.4

Potete visualizzare i modelli in diversi modi, per esempio con il Package Overview qui mostrato.

Come ho già detto, una volta che avete un modello visuale, creato da zero o generato con Reverse EngineeringWizard, potete generare automaticamente il codice basandovi sulle relazioni del vostro modello.

Per generare il codice, selezionate Code Generation *dal menu* Tools, *e seguite i passi nel Code Generation Wizard.*

Riepilogo

Il primo passo per capire dove volete andare oggi, o in qualsiasi altro giorno, consiste nel capire la natura degli strumenti che vi possono aiutare ad arrivarci. Gli strumenti per i database della Enterprise Edition non sono per i deboli. Queste sono implementazioni serie, allo stato dell'arte. Se volete usarle nelle vostre applicazioni aspettatevi di perdere del tempo per capire come funzionano. Dopo aver letto questo capitolo, dovreste avere una comprensione di base degli strumenti per i database della Enterprise Edition e di che cosa potete farci.

- Ho descritto le applicazioni Back Office.
- Ho fornito una introduzione a SQL Server.
- Avete imparato come lavorare con Microsoft Transaction Server.
- Ho spiegato Microsoft Visual Modeler.
- Vi ho insegnato come usare Reverse Engineering Wizard.

GUIDA IN LINEA

- Guida in linea HTML
- Caratteristiche della guida in linea di Windows
- Costruire una guida in linea
- RoboHelp

Alcune volte abbiamo tutti bisogno di aiuto. Non importa quanto possa essere chiara l'interfaccia di un programma, ci sarà sempre un utente che avrà bisogno della spiegazione di una caratteristica particolare. Oppure ci sarà sempre l'utente che rifiuta di leggere il manuale e farà affidamento solo sui file della guida in linea. Un sistema di guida in linea completo, sensibile al contesto è un obbligo per ogni applicazione professionale.
Creare e compilare i file della guida in linea di Windows è molto facile una volta che avete visto tutti i compiti dei suoi componenti. Questo capitolo vi guiderà alla creazione di guide in linea complete con salti, pop-up, grafica e hot spot.

Guida in linea HTML

Le guide in linea basate su HTML, che usano un browser Web per visualizzare i file della guida in linea scritti in HTML, stanno diventando velocemente un metodo standard per fornire le funzionalità della guida in linea come la documentazione e l'assistenza all'utente.

Potete scaricare HTML Help Workshop di Microsoft dall'indirizzo http://www.microsoft.com/workshop/author/htmlhelp/default.htm.

La guida in linea HTML di Microsoft viene costruita usando il controllo ActiveX guida in linea HTML. È diverso nelle caratteristiche e nel formato rispetto a NetHelp di Netscape. RoboHelp, che verrà discusso più avanti in questo capitolo, e altri prodotti di terze pani vi permettono di creare le guide in linea HTML, NetHelp, o le guide in linea Windows da un insieme difile sorgente.

Mentre questo libro va in stampa, le guide in linea HTML comprendono la maggior parte delle caratteristiche delle guide in linea Windows, ma con i file sorgente

basati su HTML multipiattaforma. Senz'altro le guide in linea HTML diventeranno il supporto prevalente per i file di aiuto, ma per il momento ogni approccio ha i suoi vantaggi e svantaggi.
Le guide in linea HTML sono veramente multipiattaforma, possono essere distribuite attraverso il Web, e possono essere estese usando tecnologie come ActiveX Java, JavaScript, VBScript, Cascading Style Sheet e altro. Ma il motore delle guide in linea Windows, Winhelp.exe, viene distribuito con ogni copia di Windows, e può essere già trovato su decine di milioni di PC in tutto il mondo. A differenza delle guide in linea HTML, che in molti casi richiedono l'installazione di un nuovo motore sul sistema, WinHelp è già presente ed è pronto per essere eseguito. Gli utenti hanno già dimestichezza con questa applicazione che peraltro è molto robusta. Un altro punto a favore del motore WinHelp è che questo è più lineare e veloce rispetto al motore HTML.

Caratteristiche delle guide in linea di Windows

La guida in linea di Windows, chiamata anche WinHelp, comprende le seguente caratteristiche:

- Una metafora con pagine a scheda permette all'utente di vedere gli argomenti nella scheda *Sommario,* di ricercare gli argomenti nella scheda *Indice,* oppure di trovare gli argomenti usando delle parole chiave nella scheda *Trova.*
- I pulsanti di collegamento permettono all'utente di muoversi direttamente a un comando dell'applicazione. Per esempio, se l'utente ha lanciato la guida in linea per trovare come installare un nuovo dispositivo hardware in Windows 95/98, un pulsante di collegamento nell'argomento sull'installazione di nuovo hardware lo porterà direttamente alla procedura guidata Nuovo hardware.
- Gli utenti possono accedere a menu sensibili al contesto in qualsiasi momento facendo clic con il pulsante destro nella finestra della guida in linea. Con questo menu, l'utente può fare un annotazione, copiare o stampare un argomento, cambiare la dimensione del carattere della finestra della guida in linea, rendere la finestra sempre visibile, oppure modificare la finestra della guida in linea con i colori di sistema (se per la guida in linea è stato usato un altro schema di colori).
- Gli utenti possono accedere all'aiuto sensibile al contesto usando il pulsante destro del mouse oppure il pulsante *?* che si trova nell'angolo superiore destro di una finestra. Questa è una finestra a scomparsa che visualizza alcune informazioni relative ad una particolare caratteristica di una finestra o di una finestra di dialogo.

Dal punto di vista dello sviluppatore, caratteristiche importanti non immediatamente visibili comprendo la capacità per:

- Creare schede personalizzate
- Impostare i caratteri e il video di default fino a colori a 24 bit
- Creare dei bitmap trasparenti il cui colore di sfondo coincide con quello della finestra in cui sono visualizzati
- Aggiungere dei pulsanti in qualsiasi punto di un argomento relativo alla guida
- Collegare i file della guida .Hlp insieme in maniera trasparente
- Posizionare le finestre secondarie in maniera assoluta
- Aggiungere file multimediali (.Avi) a qualsiasi argomento della guida. I file multimediali possono essere compilati direttamente nel file .Hlp in modo che non dobbiate includere separatamente i file con il programma di installazione.

Come creare una guida in linea

Una guida in linea, con i suoi argomenti, riferimenti, salti e pop-up, può essere complessa come un programma Visual Basic. Di conseguenza, conviene avere una solida architettura del progetto per la guida in linea prima di iniziare a creare un progetto della guida.

Pianificare un progetto per la guida in linea

Come prima cosa dovete prendere in considerazione l'applicazione per la quale state *realizzando* il progetto per la guida in linea e trovare gli argomenti della guida.

A questo punto ho visto che le vecchie schede su carta possono essere molto comode. Le uso per organizzare le informazioni, per esempio scrivendo un argomento della guida su ognuna, e poi le ordino in ordine alfabetico.

Poi dovete pianificare il progetto della guida in linea. Questo comprende:

- Definire a chi è indirizzato il vostro prodotto. È un gioco per bambini? Oppure è una applicazione per utenti esperti? Il linguaggio e i contenuti per queste due guide in linea saranno completamente diversi.
- Pianificare i contenuti del progetto della guida in linea. Questo comprende argomenti come la progettazione del menu della guida nell'applicazione, la creazione di schermate di contenuti che descrivono il programma, fino alle parole chiave e all'indice per gli argomenti.
- Creare una struttura per gli argomenti della guida.
- Decidere quali controlli nell'applicazione saranno sensibili al contesto.
- Progettare gli argomenti attuali della guida. Questo comprende scrivere gli argomenti, aggiungere la grafica e i file multimediali e l'utilizzo dei colori. (Potreste usare del testo in blu per evidenziare qualcosa, non usate troppi colori ed effetti speciali!)

Tipi di file per creare una guida in linea

Dopo che avete pianificato il progetto della guida in linea, è giunto il momento di mettersi al lavoro per creare i file che andranno nella guida in linea compilata Potete usare diversi tipi di file per creare una guida in linea di Windows compilata come elencato in Tabella 34.1.

Tabella 34.1 *Tipi di file che potete usare per creare i file della guida in linea di Windows*

Estensione del file	Tipo del file	Descrizione	Necessario per la guida?
.Rtf	Argomenti della guida	Contiene il testo per il file della guida e il codice necessario per collegare gli argomenti. Può anche contenere della grafica o delle chiamate a file grafici.	Sì
.Cnt	Sommario della guida	Contiene la numerazione gerarchica degli argomenti della guida che creano gli elementi che si trovano nella scheda Sommario quando viene eseguita la guida.	Sì
.Hpj	Progetto della guida	Simile ad un file di profilo privato con le intestazioni delle sezioni. Questo file contiene un elenco di file di testo e di grafica necessari per i file della guida, i nomi delle macro, la definizione delle finestre secondarie e le istruzioni opzionali.	Sì
.Bmp, .Wmf	Grafica	Necessario solo se fate riferimento alla grafica nel file .Rtf invece di incorporarlo.	No
.Shg	Ipergrafica	Una grafica arricchita con uno o più hot spot creati usando Shed.Exe.	No
.Mrb	Bitmap a risoluzione multipla	Uno speciale bitmap compilato da Mrbc.Exe che contiene più di una versione del bitmap a diverse risoluzioni dello schermo.	No
.Avi	Multimedia	Si fa riferimento a questi file nel file .Rtfusando{mciFileName}.	No

La guida in linea compilata in questo capitolo è creata per l'applicazione Mortgage Calcuìator, LoanCalc.Vbp, realizzata nel Capitolo 22. L'applicazione ed i suoi file associati, che comprendono diversi file di progetto della guida in linea, bitmap e il file Word Rtf usato per creare il file relativo agli argomenti della guida, sono disponibili nel CD-ROM allegato nella directory Chapter 34.

In questo capitolo creerò alcuni dei tipi di file elencati nella Tabella 34.1, nel seguente ordine:

1. File relativo degli argomenti alla guida in linea (.Rtf)
2. File ipergrafico (.Shg)
3. File di sommario (.Cnt)
4. File del progetto della guida in linea (.Hpj)

Il file relativo agli argomenti della guida sarà creato in un elaboratore di testi che supporta il Rich Text Format. L'ipergrafica sarà creata con un Hotspot Editor, Shed.Exe, e i file di sommario e del progetto della guida saranno creati con lo Help Compiler Workshop di Microsoft.

Help Compiler Workshop

Le applicazioni Help Compiler Workshop possono essere trovate nella cartella Microsoft Visual Studio\Common\Tools. Potete avviare Help Workshop usando il gruppo di programmi Microsoft Visual Studio 6.0 Tools. Diverse applicazioni sono usate nella creazione dei file della guida in linea:

- Help Workshop (Hcw.Exe) vi aiuta a creare velocemente i file del progetto (.Hpj) e di sommario.
- Help Compiler (Hcrtf.Exe) è un compilatore per la guida in linea che funziona insieme a Hcw.Exe.
- Help Author's Guide (Hcw.Hlp) è una guida in linea che fornisce informazioni complete sulla realizzazione del progetto di una guida in linea, dalla pianificazione iniziale fino alla compilazione.
- Hotspot Editor (Shed.Exe) è un editor grafico che vi aiuta a creare le bitmap con più hot spot che attivano finestre a scomparsa o dei collegamenti.
- Multiple Resolution Bitmap Compiler (Mrbc.Exe) è un compilatore che combina bitmap di diverse risoluzioni in una bitmap a risoluzione multipla.
- Dialog Box Help Editor (Dbhe.Exe) utilizza Microsoft Word for Windows per creare velocemente degli aiuti sensibili al contesto.

Una volta che avete pianificato un progetto per la guida in linea, dovete creare come prima cosa il file relativo agli argomenti della guida.

Come codificare il file relativo agli argomenti della guida

Potete creare il file relativo agli argomenti della guida in qualsiasi programma di elaborazione testi che ppossa salvare i file in Rich Text Format (.Rtf). Per la guida in linea d'esempio per questo capitolo, ho usato Word for Windows. Codificare il file relativo agli argomenti della guida è facile, una volta capito come funziona il file. Il progetto della guida in linea usa dei codici di controllo speciali per compiti specifici, alcuni dei quali sono mostrati nella Tabella 34.2.

Tabella 34.2 *Codici di controllo usati nel file di progetto della guida in linea.*

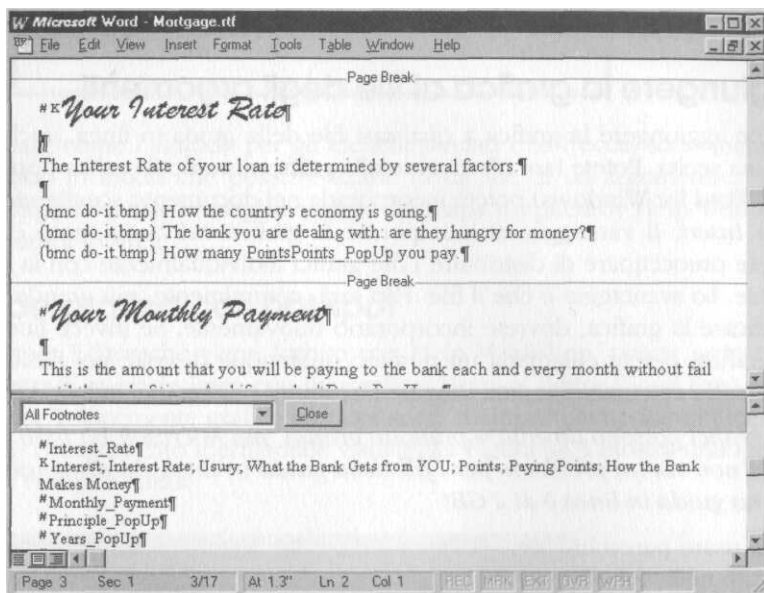
Codice di controllo	Nome formale	Descrizione
# (footnote)	Stringa di contesto	Definisce una stringa di contesto che identifica univocamente un argomento relativo alla guida
$ (footnote)	Titolo	Definisce il titolo dell'argomento relativo alla guida
K (footnote)	Parola chiave	Definisce una parola chiave che l'utente utilizza quando cerca un argomento
+ (footnote)	Numero di sequenza	Definisce una sequenza che determina in quale ordine l'utente può sfogliare gli argomenti (opzionale)
* (footnote)	Tag di build	Definisce un tag che specifica gli argomenti compilati condizionalmente dal compilatore della guida in linea
Testo con doppia sottolineatura o barrato	Salto	Appare nel file della guida in linea compilato come testo in verde, con sottolineatura singola, indicando all'utente che può fare clic per saltare ad un altro argomento.
Testo con singola sottolineatura	Definizione (pop-up)	Viene mostrato nel file della guida in linea compilato come testo in verde, con sottolineatura non continua. Quando l'utente fa clic sul testo, oppure preme Invio, apparirà una finestra pop-up.
Testo nascosto	Stringa di contesto	Specifica la stringa di contesto che appare per l'argomento quando l'utente fa clic nel testo che lo precede.

Per un elenco completo dei codici di controllo, aprite "Topic footnotes" nella Help Author's Guide. Usando la Tabella 34.2 come riferimento, guardate la Figura 34.1. Questa figura mostra il file relativo agli argomenti della guida Mortgage.Rtf (oppure potete aprire il file che si trova nel CD-ROM allegato).

La Figura 34.1 mostra due argomenti della guida: "Your Interest Rate" e "Your Monthly Payment". Gli argomenti devono essere divisi da interruzioni di pagina, ed ogni argomento deve avere una unica stringa di contesto.

Ecco come creare un argomento con una stringa di contesto. All'inizio della riga inserite una *footnote* (nota a pie di pagina) con il carattere #. Nell'area delle note, digitate il nome della stringa di contesto. Nel caso dell'argomento "Your Interest Rate", la stringa di contesto è `Interest_Rate`. Ritornate alla finestra principale e digitate l'argomento della guida. Questo è tutto.

Se volete aggiungere un titolo all'argomento della guida, sposta il punto di inserimento subito dopo il simbolo di nota # e aggiungete una nota con il carattere $. Nella sezione delle note, immettete un titolo, come *Mortgage: Interest Rate.*

Figura 34.1
Ecco il file relativo agli argomenti della guida Mortgage.Rtf in Word for Windows, che mostra gli argomenti della guida nella parte superiore della finestra e i codici di controllo nella pane inferiore.

Per aggiungere le parole chiave che faranno riferimento all'argomento "Your Interest Rate", mettete il punto di inserimento subito dopo il simbolo di nota # e aggiungete un simbolo di nota k. Nella sezione delle note, come mostrato nella Figura 34.1, immettete le parole chiave in questo modo:

```
Kinterest; Interest Rate; Usury
```

Notate che le parole chiave sono separate da punti e virgola.

Collegare gli argomenti con i salti e con i pop-up

Una volta che avete inserito gli argomenti della guida, potete aggiungere i salti e i pop-up. I salti sono mostrati da elementi in testo verde, con sottolineatura singola che appare nel file della guida compilato. Facendo clic su un salto, l'utente verrà portato a un altro argomento relativo. Pop-up (o definizioni) sono in testo verde, sottolineato con riga non continua nei file della guida. Facendo clic su un testo di

questo tipo, oppure premendo Invio, verrà mostrata una finestra pop-up contenente una definizione del testo sottolineato.
Cr'eate i salti ed i pop-up usando il testo nascosto e le sottolineature. Per creare un salto, selezionate la parola o le parole che volete usare per indicare il salto ed evidenziatelo con una sottolineatura doppia o barratelo. Subito dopo il testo (non aggiungete nessuno spazio!), modificate il carattere in Nascosto, e digitate il nome del contesto senza usare gli spazi. (Il centro della Figura 34.1 contiene la stringa di contesto `Points_PopUp`. Notate che è sottolineata con riga non continua, che è il modo in cui Word indica che il testo è nascosto.)
Per creare un pop-up, selezionate la parola o le parole che volete usare per indicare il pop-up e sottolineate il testo. Mettete il nome del contesto nascosto subito dopo le parole sottolineate come avete fatto per i salti. Il centro della Figura 34.1 contiene il pop-up `Points`, con la sua stringa di contesto nascosta `Points_PopUp` alla destra.

Aggiungere la grafica ai file degli argomenti

È facile aggiungere la grafica a qualsiasi file della guida in linea, anche se dovete fare una scelta. Potete fare riferimento alla grafica usando il codice oppure (quando usate Word for Windows) potete incorporarla nel documento scegliendo *Picture* dal menu *Insert.* Il vantaggio di incorporare la grafica nel documento è che non vi dovrete preoccupare di distribuire i file grafici individualmente con la vostra applicazione. Lo svantaggio è che il file .Hlp sarà, naturalmente, più grande, e se dovete modificare la grafica, dovrete incorporarlo nuovamente. Se invece fate riferimento alla grafica, potete chiamarla un numero qualunque di volte nel file della guida e modificarlo esternamente, ma dovrete distribuire i file grafici individualmente.

I file grafici possono diventare piuttosto grandi, ma Microsoft ha fatto in modo che questo non sia un problema per i file della guida in linea. La dimensione massima per una guida in linea è di 2 GB!

Potete usare parecchi comandi quando fate riferimento ai file .Bmp o .Wmf, come mostrato nella Tabella 34.3.

Tabella 34.3 *Comandi per far riferimento ai file grafici nei testi degli argomenti detta guida.*

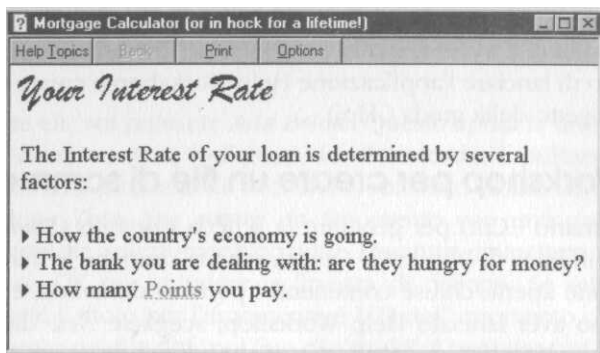
Comando	Descrizione
bmc	Allinea la grafica come un carattere sulla linea base del carattere
bml	Allinea la grafica al margine sinistro e spezza il testo al lato destro della grafica
bmr	Allinea la grafica al margine destro e sagoma il testo lungo il lato sinistro della grafica

La Figura 34.1 mostra tre riferimenti al file Do-it.Bmp usando il comando `bmc`. La Figura 34.2 mostra l'aspetto finale dell'argomento della guida.

Potete anche aggiungere un parametro `t` *ad ogni comando elencato in Tabella 34.3. Per esempio:*

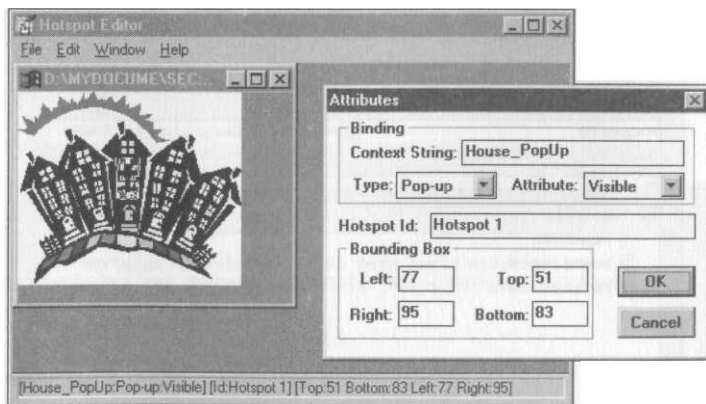
```
{bmlt MyGraphic.Bmp}
```

La t crea una bitmap trasparente, indicando a Help Workshop di sostituire il colore dello sfondo della bitmap con il colore di sfondo della finestra in cui verrà visualizzata. Questo comando però funziona solo con bitmap a 16 colori.

Figura 34.2 *Questo argomento della guida fa riferimento tre volte a una bitmap triangolare usando il comando bmc.*

Diverse bitmap, come i simboli per gli elenchi puntati e le frecce, sono incluse in Help Workshop in modo che possiate usarle in un file di un argomento. Per un elenco completo di queste immagini, vedere "Bitmaps supplied by Help Workshop" nella Help Author's Guide.

Creare ipergrafica con hot spot

Usando Hotspot Editor che viene fornito con Help Workshop, potete arricchire il vostro progetto di guida in linea con ipergrafica. Hotspot Editor (Shed.Exe) vi permette di creare velocemente grafica con hot spot, chiamata anche ipergrafica. Uno hot spot è un collegamento ipermediale visuale. La Figura 34.3 mostra una ipergrafica creata in Hotspot Editor.

Figura 34.3 *Potete usare Hotspot Editor, Shed.Exe, per creare ipergrafica, grafica con hot spot.*

Per creare una ipergrafica, lanciate Hotspot Editor e scegliete *Open* dal menu *File* per caricare un file grafico. Usate il puntatore del mouse per disegnare un rettangolo dove volete che appaia l'hot spot. Quando lo fate, verrà aperta una finestra di dialogo *Attributes*. Immettete la stringa di contesto (creata nel file relativo agli argomenti

della guida) a cui questo hot spot sarà collegato e impostate il tipo di hot spot che volete: salto o pop-up. Fate clic su *OK per* chiudere la finestra di dialogo *Attributes,* e poi scegliete *Save As* dal menu *File* per salvare l'ipergrafica come file .Shg.
Potete fare riferimento alla ipergrafica nei file relativi agli argomenti della guida come fareste per qualsiasi altra bitmap usando uno dei comandi elencati nella Tabella 34.3. Una volta che avete finito la creazione del file relativo agli argomenti della guida, è tempo di lanciare l'applicazione Help Workshop e creare i file di sommario (.Cnt) e il progetto della guida (.Hpj).

Uso di Help Workshop per creare un file di sommario

Usate il file di sommario (.Cnt) per generare la scheda *Contents (Sommario)* nella finestra della guida in linea di Windows. Questa scheda contiene le immagine grafiche, i libri e le pagine aperte/chiuse contenenti i punti di domanda, e gli elementi del sommario. Dopo aver lanciato Help Workshop, scegliete *New* dal menu *File.* Verrà aperta una finestra di dialogo *New,* che vi permetterà di creare i file del progetto della guida o il sommario della guida, come mostrato nella Figura 34.4.

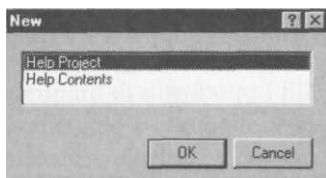
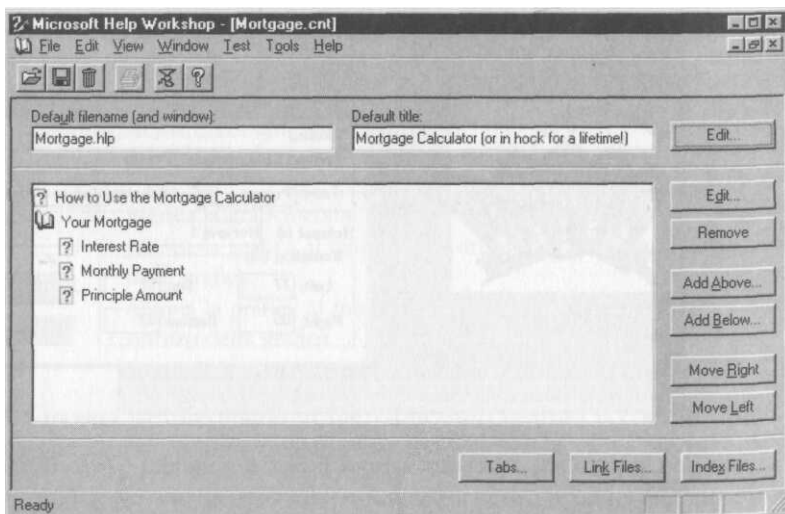
Figura 34.4 *Creazione di un nuovo file del progetto per la guida in linea.*

Selezionate *Help Contents* e poi fate clic su *OK.* Help Workshop aprirà una schermata per il file di sommario come mostrato in Figura 34.5 (ma senza nessun elemento).

Figura 34.5 *La schermata del file di sommario che Help Workshop apre aiuta a creare velocemente un file di sommano.*

Per il file di sommario, iniziate nella parte alta della finestra e fate clic sul pulsante *Edit* alla destra del campo *Default Title.* Verrà aperta una finestra di dialogo *Default Help Information.* In questa finestra di dialogo, immettete:

- Il nome di default per la vostra guida in linea, per esempio Mortgage.Hlp
- Il titolo di default per la vostra guida in linea, per esempio, Mortgage Calculator

Di seguito, fate clic sul pulsante *AddBelow.* Questo aprirà la finestra di dialogo *Edit Contents Tab Entry* mostrata in Figura 34.6. Selezionate il pulsante di opzione *Heading* per creare una intestazione con una icona libro vicino ad essa oppure il pulsante di opzione *Topic* per creare un argomento con una pagina con punto di domando vicino. Se selezionate l'opzione *Heading,* immettete una intestazione e poi fate clic su *OK* per chiudere la finestra di dialogo. Se selezionate l'opzione *Topic,* immettete il titolo per l'argomento e l'ID dell'argomento che avete assegnato a quell'argomento nel file .Rtf, poi fate clic su *OK.* Continuate ad aggiungere le intestazioni e gli argomenti fino a quando non avete aggiunto tutti gli elementi necessari per la vostra guida in linea.

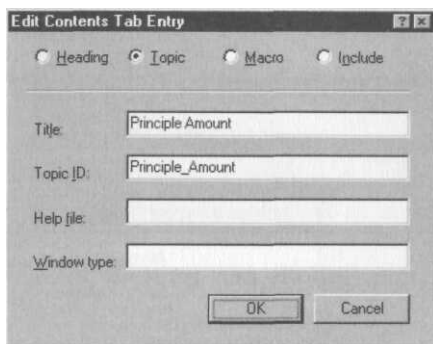
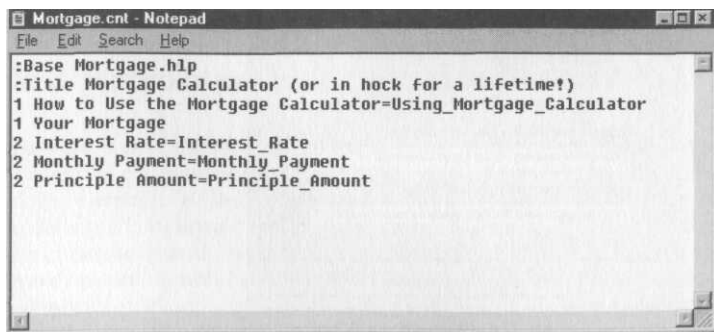
Figura 34.6
La finestra di dialogo Edit Contents Tab Entry vi permette di immettere velocemente gli argomenti e le intestazioni della scheda Contents.

Un file di sommario è semplicemente un file di puro testo salvato con l'estensione .Cnt. Se aprite il file di sommario Mortgage.Cnt creato con Help Workshop, vedrete quello che è mostrato nella Figura 34.7.

Figura 34.7
Il file di sommano per la guida in linea d'esempio, Motgage.Cnt, imposta gli elementi del sommano che si trovano nella scheda Contents quando Mortgage.Hlp è compilato ed eseguito.

Mortgage.cnt - Notepad

File Edit Search Help

```
:Base Mortgage.hlp
:Title Mortgage Calculator (or in hock for a lifetime!)
1 How to Use the Mortgage Calculator=Using_Mortgage_Calculator
1 Your Mortgage
2 Interest Rate=Interest_Rate
2 Monthly Payment=Monthly_Payment
2 Principle Amount=Principle_Amount
```

Creare il file del progetto della guida in linea

Quando avete finito di aggiungere gli elementi al file di sommario, salvatelo e scegliete *New* dal menu *File* per aprire la finestra di dialogo *New* e creare il file del progetto della guida in linea (.Hpj). Dopo che avete selezionato *Help Project* nella finestra di dialogo *New,* l'interfaccia di Help Workshop cambierà per aiutarvi a creare il file .Hpj, come mostrato nella Figura 34.8. (La finestra di Help Workshop per un nuovo file .Hpj sarebbe vuota.)

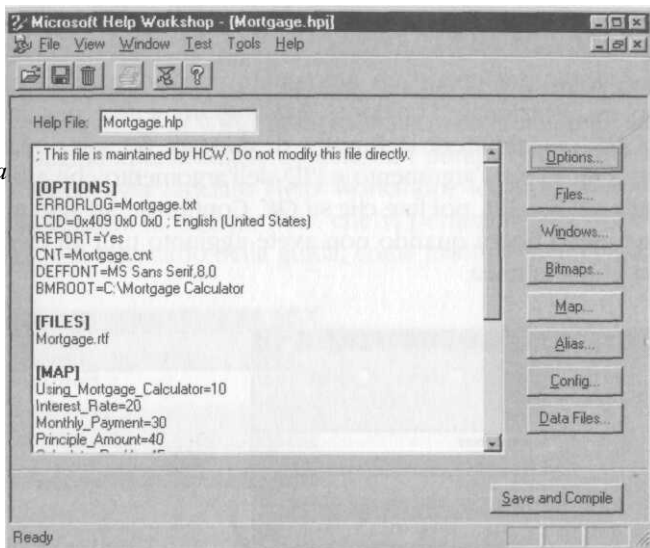
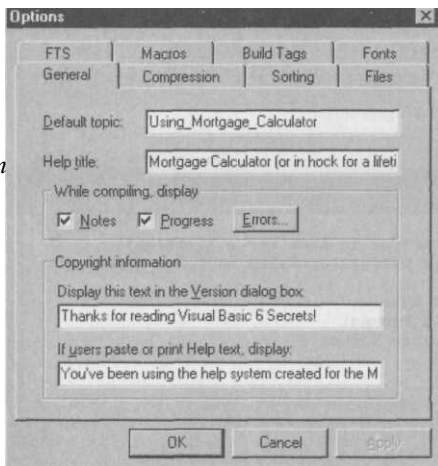
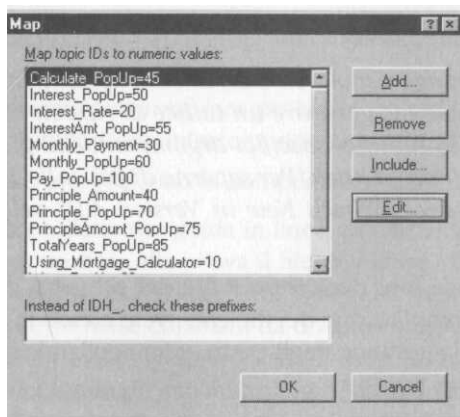
Figura 34.8 *L'interfaccia utente di Help Workshop può sembrare complicata all'inizio, ma una volta che avrete creato i file del progetto della guida e di sommario, diventerà più chiara.*

Per continuare, fate clic sul pulsante *Options* per aprire la finestra di dialogo *Options* mostrata nella Figura 34.9.

Figura 34.9 *Usate la finestra di dialogo Options per impostare i necessari parametri per un file del progetto della guida.*

Dovrete impostare diversi elementi nelle varie schede della finestra di dialogo *Options* per costruire il file del progetto della guida in linea:

- Nella scheda *General* impostate l'argomento di default, il titolo per la guida e le informazioni sul copyright per la guida.
- Nella scheda *Files,* impostate il nome di default per la guida (per esempio Mortgage.Hlp), il nome e la posizione della cartella del file di testo per il log degli errori (per esempio, Mortgage.Txt), e la posizione del file relativo agli argomenti della guida (.Rtf) e del file di sommario (.Cnt) che avete creato in precedenza.
- Nella scheda *Fonts,* impostate il carattere di default per le finestre della vostra guida in linea.
- Nella scheda *Compression,* impostate la quantità di compressione, nessuna, massima o personalizzata, che sarà applicata alla guida in linea.

Di seguito, fate clic sul pulsante *Bitmaps* per aprire la finestra di dialogo *Bitmap Folders.* Usate il pulsante *Browse* in questa finestra di dialogo per indicare a Help Workshop deve si trovano le bitmap a cui si fa riferimento nel file relativo agli argomenti della guida (.Rtf). Il pulsante successivo su cui fare clic è *Map.* Questo aprirà la finestra di dialogo *Map* mostrata nella Figura 34.10. Usate questa finestra di dialogo per impostare gli ID degli argomenti, che avete creato nel file degli argomenti, ad un numero intero positivo. Per esempio, nella Figura 34.10, Interest_Rate è uguale a 20.

Figura 34.10

Usate la finestra di dialogo Map per assegnare dei valori interi come ID degli argomenti. Questi numeri sono usati per chiamare la guida in linea sensibile al contesto direttamente da una applicazione.

Aggiunta di macro al progetto della guida in linea

Potete aggiungere, a un progetto della guida in linea, macro che eseguano molti compiti, come l'aggiunta e la rimozione di pulsanti e menu personalizzati, la modifica delle funzioni degli elementi e dei pulsanti del menu, l'esecuzione di applicazioni dall'interno della guida in linea e per eseguire della chiamate a .Dll. Le macro possono anche essere chiamate quando la guida in linea viene aperta, quando l'utente seleziona un argomento, oppure da uno hot spot. Parecchie macro pronte all'uso sono già integrate nel compilatore della guida. Per un elenco completo di queste macro, consultate l'argomento "Macro quick reference" nella Help Author's Guide.

Proprio come il file di sommario creato in precedenza, il file del progetto della guida in linea è un file di semplice testo che potete visualizzare in qualsiasi editor di testo, come mostrato nella Figura 34.11.

Figura 34.11
il file del progetto detta guida in linea creato in Help Workshop è un file di puro testo che può essere visualizzato in un editor di testo.

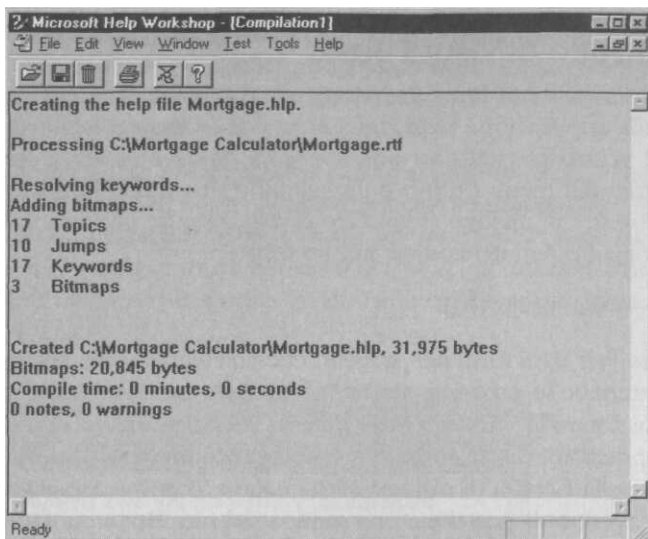
```
Mortgage.hpj - Notepad
File Edit Search Help
; This file is maintained by HCW. Do not modify this file directly.

[OPTIONS]
ERRORLOG=Mortgage.txt
LCID=0x409 0x0 0x0 ; English (United States)
REPORT=Yes
CNT=Mortgage.cnt
DEFFONT=MS Sans Serif,8,0
BMROOT=C:\Mortgage Calculator
HLP=Mortgage.hlp

[FILES]
Mortgage.rtf

[MAP]
Using_Mortgage_Calculator=10
Interest_Rate=20
Monthly_Payment=30
Principle_Amount=40
Calculate_PopUp=45
Interest_PopUp=50
InterestAmt_PopUp=55
Monthly_PopUp=60
Principle_PopUp=70
```

Con Help Workshop, potete aggiungere molte caratteristiche supplementari a un progetto della guida in linea, per esempio generare un indice di ricerca testuale, impostare delle compilazioni condizionali per specifici argomenti della guida e creare versioni per altre lingue della guida in linea. Per saperne di più sulle caratteristiche complete di Help Workshop, vedere "What's New in Version 4.0" nell'indice detta Help Author's Guide.

Questo è tutto quello che dovete fare per creare il file del progetto della guida in linea. Il passo successivo è la compilazione dei file che avete creato in un unico file della guida (.Hlp), predisporre la gestione degli errori, e fare collaudare automaticamente a Help Workshop i salti e i pop-up.

Compilazione e collaudo della guida in linea

Ora che tutti i componenti sono pronti per questa guida in linea di Mortgage Calculator, compiliamolo! Compilare una guida in linea con Help Workshop è molto più facile rispetto alla vecchia compilazione sulla riga di comando a cui siete abituati. Fate semplicemente clic sul pulsante *Save and Compile* nell'angolo in basso a destra della finestra di Help Workshop (si veda la Figura 34.8). Quando Help Workshop ha completato la compilazione, aprirà una schermata di log che descrive il file che è stato compilato ed evidenzia se si sono verificati degli errori, come mostrato nella Figura 34.12.

Figura 34.12 *Quando un progetto della guida in linea è compilato, Help Workshop visualizza un log che mostra le informazioni sulla compilazione ed ogni errore che si è verificato.*

Help Workshop offre tre tipi di messaggi di errore, quando compila una guida in linea:

- Note per i problemi che non influenzano il modo in cui funziona la guida in linea
- Avvertimenti *(warning)* per problemi trovati in un file di guida, anche se questi problemi non interrompono la compilazione
- Errori che impediscono la compilazione del file

Dopo aver compilato una guida in linea, potete usare le voci del menu Test di Help Workshop per mettere alla prova il file di sommario e tutti i pop-up e i salti, inviare macro per vedere come verranno eseguite nella guida, e chiamare una API WinHelp come se fosse chiamata da un altro programma.
Un'altra caratteristica di Help Workshop, Help Author, fornisce informazioni supplementari sul debugging e sulPauthoring. Quando viene selezionato Help Author nel menu *File:*

- Il titolo della finestra della guida mostra il numero dell'argomento invece del titolo della guida.
- Potete premere Ctrl+Maiusc+freccia destra o sinistra per avanzare attraverso gli argomenti della vostra guida.
- Sono visualizzate specifiche informazioni sui problemi nel progetto della guida in linea oppure nei file dei contenuti.
- Potete fare clic con il pulsante destro su qualsiasi argomento della guida per visualizzare informazioni sull'argomento corrente, oppure potete fare clic con il pulsante sinistro su qualsiasi hot spot per visualizzare l'ID dell'argomento dell'hot spot oppure qualsiasi macro associata con l'hot spot.

Collegamento di un file di guida con un progetto di Visual Basic

La creazione della guida in linea per Mortgage Calculator ha richiesto del tempo ma collegarla alla applicazione Mortgage Calculator è veloce e facile. Il primo passò è connettere il vostro progetto ad una specifica guida in linea. Potete farlo scegliendo *Properties* dal menu *Project* e immettendo una guida in linea nella scheda *General* della finestra di dialogo *Project Properties,* oppure impostando la proprietà HelpFile dell'oggetto App nel codice, per esempio:

```
App.HelpFile = "Mortgage.Hlp"
```

Successivamente, per ogni form nel progetto che volete collegare, selezionate il form e modificate entrambe le proprietà WhatsThisButton e WhatsThisHelp a True. Poi selezionate ogni controllo che avrà una guida in linea sensibile al contesto ed impostate la sua proprietà WhatsThisHelpID all'appropriato numero ID del contesto che è stato impostato nella finestra di dialogo *Map* quando Mortgage.Hpj è stato creato. (È molto comodo avere a disposizione una stampa del file .Hpj a cui fare riferimento.) Per esempio, se selezionate lblPrinciple, dovreste impostare la sua proprietà WhatsThisHelpID a 70, il numero ID del contesto per Principle_PopUp. Dopo che avrete impostato tutte le proprietà WhatsThisHelpID dei controlli, è tempo di aggiungere il codice che attiva la guida in linea sensibile al contesto quando l'utente fa clic con il pulsante destro sul controllo oppure usa il pulsante *What's This?* nell'angolo in alto a destra della finestra. Dovrete usare l'evento MouseDown per i diversi controlli, controllare che il pulsante destro sia stato premuto e usare il metodo ShowWhatsThis. Però, invece di digitare *controllo.*ShowWhatsThis all'interno dell'evento MouseDown per ogni controllo, potete creare una procedura privata, GetHelp, che usa il nome del controllo come parametro e viene passata al metodo ShowWhatsThis. Per esempio, ecco il codice per l'evento lblPrinciple MouseDown e per GetHelp:

```
Private Sub lblPrinciple_MouseDown(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    If Button = vbRightButton Then
        GetHelp lblPrinciple
    End If
End Sub

Private Sub GetHelp(C As Control)
    C.ShowWhatsThis
End Sub
```

Un tocco finale per questa applicazione sarebbe l'aggiunta di un piccolo menu che contenga un titolo "Help" e la voce "How to Use the Calculator". Per attivare questa voce, dovete trascinare una finestra di dialogo controllo comune nel form e aggiungere le seguenti linee di codice all'evento Click della voce di menu:

```
Private Sub mnuCalcHelp_Click()
    CommonDialog1.HelpFile = "Mortgage.Hlp"
    CommonDialog1.HelpCommand = cdlHelpContents
    CommonDialog1.ShowHelp
End Sub
```

E questo è tutto! Tutto quello che dovete fare è compilare un eseguibile di questo progetto ed eseguire l'applicazione dal file .Exe. Questo perché, quando state lavorando su un programma in modalità progettazione ed eseguite un progetto, il progetto cercherà il file .Hlp nella directory corrente, Visual Basic, invece che nella directory propria del progetto.

Come regola generale, la maggiorparte delle guide in linea di Windows si trovano nella directory dell'applicazione o nella directory Windows\Help. Però, il Capitolo 16 ha mostrato una routine che ricerca un file specifico. Potrete usare questa routine per trovare la guida in linea associata con una applicazione. La guida in linea per Mortgage Calculator a questo punto è completamente funzionante, come mostrato in Figura 34.13.

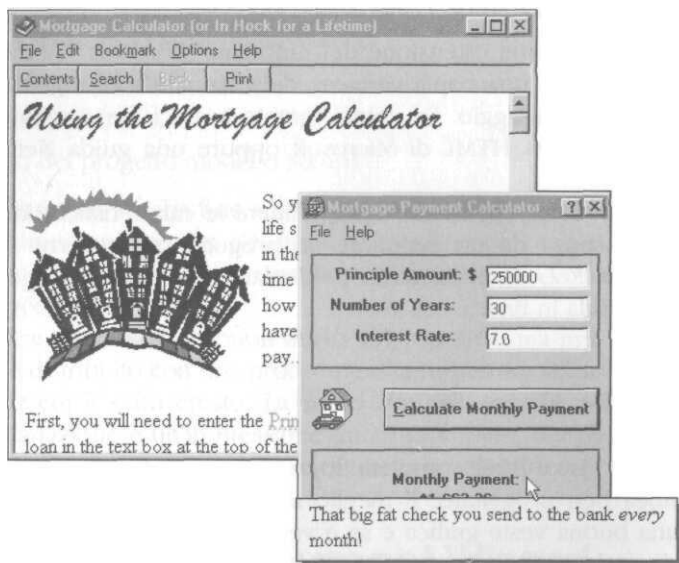
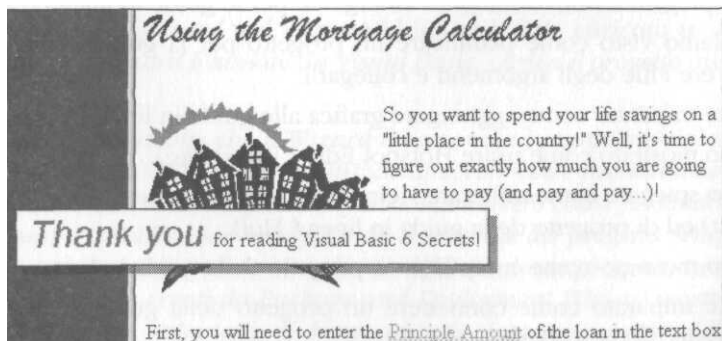
Figura 34.13 *La guida in linea per Mortgage Calculator sembra veramente professionale.*

Se fate clic sulla ipergrafica che abbiamo creato in precedenza, potete vedere che il pop-up funziona correttamente come mostrato nella Figura 34.14.

Figura 34.14 *L'hot spot dell'ipergrafica visualizza un pop-up di ringraziamento a seguito di un clic.*

Strumenti per la guida in linea di terze parti: RoboHelp

Sono disponibili sul mercato numerosi prodotti che rendono molto più facile la creazione delle guide in linea e la loro connessione ai progetti di Visual Basic. Il prodotto principale in questa categoria è RoboHelp della Blue Sky Software Corporation. Come forse avrete notato, la creazione delle guide in linee è diventata una specializzazione professionale. Pochi creatori di guide in linee professionisti tenterebbero di creare un sistema di guida in linea senza usare RoboHelp (oppure uno degli altri strumenti professionali paragonabili).

RoboHelp è un prodotto di Blue Sky Software. Potete trovare maggiori informazioni al sito Web della Blue Sky a http://www.blue-sky.com.

RoboHelp è costruito come estensione dell'ambiente Word for Windows. Usando RoboHelp potete creare una copia cartacea del manuale e una guida in linea di Windows in un solo passaggio. Se volete, potete usare i file sorgente per creare anche una guida in linea HTML di Microsoft oppure una guida NetHelp di Netscape.
Essenzialmente, RoboHelp permette di aggiungere le caratteristiche della guida in linea visualmente senza dover compilare un progetto per vederne l'aspetto. Gli strumenti inclusi in RoboHelp rendono particolarmente facile collegare i progetti Visual Basic alla guida in linea.

Riepilogo

Ogni applicazione Visual Basic completa dovrebbe essere distribuita con un sistema di guida in linea ben progettato. E meglio ancora il sistema di guida in linea è vivace, con una buona veste grafica e se è veramente d'aiuto. Capire come creare una guida in linea la prima volta non è facile. Questo è stato lo scopo del capitolo: presentare il processo di creazione di una guida in linea completa chiaramente, un passo per volta. Una volta che avrete capito il processo, troverete facile creare le guide in linea e collegare i relativi file ai progetti Visual Basic.

- Ho spiegato il nuovo standard emergente per le guide in linea HTML.
- Abbiamo visto come pianificare un progetto per la guida in linea, come scrivere i file degli argomenti e collegarli.
- Avete imparato come aggiungere grafica alle guide in linea.
- Vi ho mostrato come usare Hotspot Editor (Shed.Exe).
- Vi ho spiegato come utilizzare Help Workshop per creare i file di sommario (.Cnt) ed di progetto della guida in linea (.Hpj).
- Vi ho mostrato come compilare un progetto della guida in linea.
- Avete imparato come connettere un progetto della guida in linea ad un progetto di Visual Basic.

PROGRAMMI D'INSTALLAZIONE

- Uso di Package and Deployment Wizard
- File delle dipendenze
- Installazioni via Internet
- All'interno del progetto modello Setup1

Tutte le cose arrivano ad una fine, anche questo libro. E poiché, alla fine, abbiamo fatto un cerchio completo, non c'è momento migliore del capitolo finale per parlare dell'inizio. Cioè del vostro programma *d'installazione* (chiamato anche programma o procedura di *setup).*
Per dirla in altre parole, da un buon inizio arriva una buona fine. Se il vostro programma non è distribuito con una procedura d'installazione affidabile, non svolgerà il compito per cui è stato creato. La prima impressione che offre il vostro programma d'installazione è un'impressione duratura. Inoltre, dei problemi nell'installazione possono minare le funzionalità dell'intero pacchetto.

Package and Deployment Wizard

Package and Deployment Wizard è uno strumento separato che viene avviato dal gruppo Visual Studio 6.0 Tools dal menu *Start* di Windows. Il suo scopo è quello di guidare nel processo di creazione di procedure d'installazione.

Potete anche avviare il Wizard dal menu Add-Ins *se è stato caricato in Add-Ins Manager. Quando il Wizard è avviato da Visual Basic, carica il progetto attivo corrente.*

Il programma di installazione che il Wizard crea per voi è compilato dal progetto modello Setup, \Wizards\PDWizard\5etupl\Setup1.Vbp, che è installato nella cartella Visual Studio. Più avanti in questo capitolo, vi mostrerò come personalizzare i vostri programmi d'installazione modificando il sorgente del progetto Setupl.Vbp. Fate attenzione, però, perché una volta che modificate questo progetto, tutti i programmi di installazione creati da Package and Deployment Wizard mostreranno le modifiche che avete apportato. È bene, quindi, fare una copia dell'intera directory contenente Setupl.Vbp prima di modificare il progetto.

Usando il Package and Deployment Wizard fornito con Visual Basic, potrete preparare facilmente dei programmi d'installazione adeguati per molte situazioni. Se il programma d'installazione di default creato da Package and Deployment Wizard non soddisfa le vostre esigenze, potete personalizzare il progetto Setup1 di Visual Basic che è usato dal Wizard.

Sul mercato sono disponibili anche parecchi eccellenti strumenti di generazione prodotti da terzi, facili da usare e che creano superbi programmi d'installazione personalizzati. InstallShield, della InstallShield Corporation, è considerata la più importante utility d'installazione industriale. Un programma come InstallShield richiede un tempo di apprendimento notevole, in quanto usa il proprio linguaggio di scripting basato sul C, ma se dovete creare molti programmi professionali d'installazione, è consigliabile imparare ad usarlo. Maggiori informazioni su InstallShieldpossono essere trovate a http://www.installshield.com.

Dettagli dell'installazione gestiti dal Package and Deployment Wizard

Il Package and Deployment Wizard analizza un file di progetto (.Vbp) specificato e richiede alcune informazioni:

- Se il Wizard deve creare un programma d'installazione, se deve preparare un pacchetto e predisporlo per lo scaricamento da Internet, o se deve semplicemente creare un file delle dipendenze
- Quali file devono essere distribuiti con il pacchetto
- I driver di accesso ai dati che devono essere inclusi nel progetto
- La destinazione e il supporto per i file d'installazione
- I componenti ActiveX richiesti dal pacchetto e dalla distribuzione
- I server Remote ActiveX richiesti dal pacchetto e dalla distribuzione
- Se il pacchetto e la distribuzione devono essere installati come eseguibile, come componente ActiveX, come documento ActiveX, oppure come controllo ActiveX

Tutte le distribuzioni, non solo le distribuzioni per Internet come nel passato, sono fatte usando file compressi nel formato .Cab.

Una volta che le informazioni sono state ottenute, il Package and Deployment Wizard esegue le seguenti operazioni:

- Prepara un file di testo, Setup.Lst, che elenca tutti i file richiesti dal pacchetto e e dall'installazione, ivi compresi i componenti ActiveX, i controlli, le DLL e così via. Questo file è in un formato di profilo privato. In altre parole, è un file .Ini con le intestazioni e le parole chiave. Le informazioni possono essere quindi recuperate dal progetto Setup1, che lo utilizza per copiare i file nelle posizioni corrette, per registrare i componenti ActiveX e i controlli come richiesto, e che crea i collegamenti e le voci opportune nel menu *Start.*

- Crea un file delle dipendenze (.Dep), se quella opzione è stata selezionata (vedere "File delle dipendenze" più avanti nel capitolo).
- Comprime i file del pacchetto e di installazione e calcola il numero di dischi richiesti per la distribuzione.
- Per le installazioni per Internet, crea un pacchetto Internet e codice HTML di esempio.
- Copia i file di *avvio,* necessari per eseguire inizialmente il programma di setup, alla destinazione (per esempio sul supporto di distribuzione). Copia anche i file non compressi richiesti dal pacchetto e dall'installazione alla destinazione (e li divide su più dischi se richiesto).

Notate che se create la vostra procedura d'installazione, oppure se modificate il progetto Setup1 usato da Package and Deployment Wizard come verrà presto descritto, dovete assicurarvi di eseguire ogni passo (se questi si applicano al vostro pacchetto).
Quando la procedura di installazione predisposta viene effettivamente eseguita sulla macchina bersaglio, per esempio da un disco floppy, viene eseguito il primo file di avvio, Setup.Exe, seguito da altri programmi inclusi nella sezione [Bootstrap] di Setup.Lst. Questi file comprendono i file di Visual Basic necessari per eseguire Setup.Exe (che è esso stesso un programma Visual Basic) e Setup1.Exe, che è compilato dal progetto Setup1 e personalizzato per lo specifico pacchetto d'installazione da Package and Deployment Wizard. (Notate che è possibile cambiare il nome di questo file, purché ci sia un riferimento al nuovo nome in Setup.Lst.)

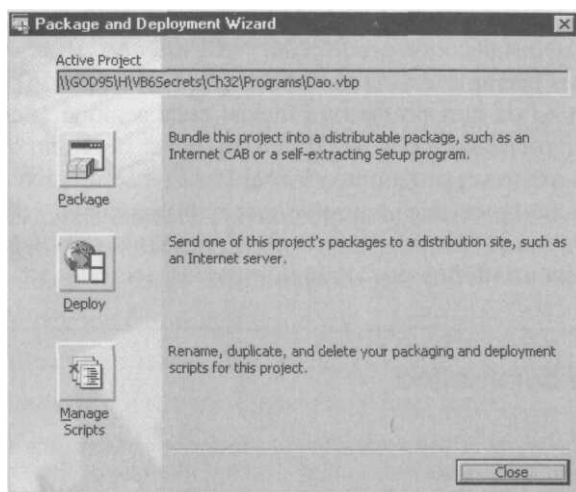
Disinstallazione automatica

Ci si aspetta che i programmi di Windows offrano la rimozione automatica come parte della loro installazione. St6unst.Exe è una utility di di installazione dei pacchetti che fa parte di Visual Basic Setup Toolkit e che è inclusa con il supporto di distribuzione da Package Deployment Wizard, che la copia nella directory Windows delle macchina bersaglio.
Il programma di avvio dell'installazione e Setupl preparano un file di log che contiene informazioni complete sulle modifiche che l'installazione ha apportato al sistema bersaglio. Se l'installazione ha avuto successo, il pacchetto viene aggiunto all'elenco dei programmi nella utility *Add/Remove (Installazione applicazioni)* del Pannello di controllo. Se si chiede la disinstallazione del programma, St6unst.Exe usa le voci del file di log per tentare di eliminare le modifiche fatte dai programmi di installazione.

L'eseguibile di avvio, Setup.Exe, è copiato nella directory Windows del bersaglio Setup.Exe poi legge le informazioni di configurazione dalla copia di Setup.Lst che è stata messa nella directory Windows. Normalmente il passo successivo consiste nel richiamare Setup1.Exe che procede con la copia dei file del pacchetto e della distribuzione in una specifica destinazione indicata dall'utente.

Punta e vai: Package and Deployment Wizard

Il funzionamento del Package and Deployment Wizard è veramente semplice. Iniziate selezionando Package and Deployment Wizard dal gruppo di progetti Visual Basici nel menu *Start* di Windows. Dopo un pannello introduttivo, vi verrà richiesto di indicare al Wizard il progetto per il quale volete preparare un programma d'installazione, come mostrato nella Figura 35.1.

Figura 35.1 *Potete usare il primo pannello del Package and Deployment Wizard per selezionare un file di progetto Visual Basic (. Vbp) e per determinare il tipo di programma d'installazione che il Wizard creerà.*

Se avete un progetto Visual Basic aperto e avviate il Wizard dal menu Add-Ins, *quel progetto sarà selezionato automaticamente.*

Potete anche usare questo pannello per scegliere tra un programma di installazione standard, una installazione via Internet oppure per creare semplicemente un file delle dipendenze.

Troverete informazioni sulla creazione di un pacchetto per il Web nel Capitolo 27.

Package and Deployment Wizard vi guida attraverso molti pannelli, ognuno dei quali richiede delle informazioni necessarie per preparare il programma d'installazione. Un pannello, mostrato nella Figura 35.2, vi permette di aggiungere manualmente i file necessari per il pacchetto e l'installazione (oppure per eliminare manualmente i file che credete che il Wizard abbia aggiunto per sbaglio).

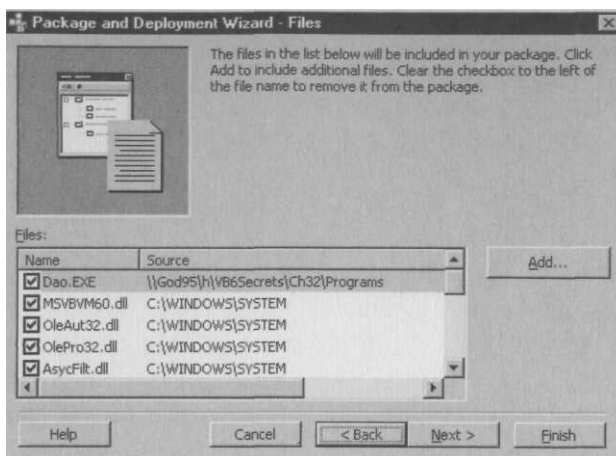
Figura 35.2
Usate un pannello del package and Deployment Wizard per aggiungere e per eliminare i file necessari per la distribuzione.

Se il vostro programma richiede l'accesso a database, il Wizard recupera le informazioni per sapere quale file di supporto distribuire, come mostrato nella Figura 35.3. L'ultimo pannello del Wizard (vedere Figura 35.4) permette di salvare un file Script sulla base delle informazioni fornite al Wizard.

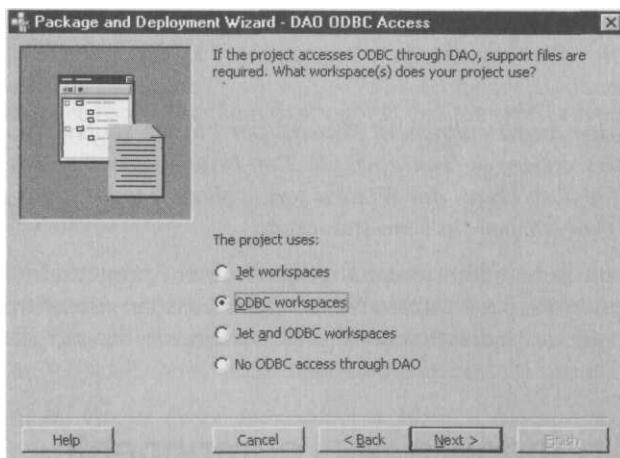
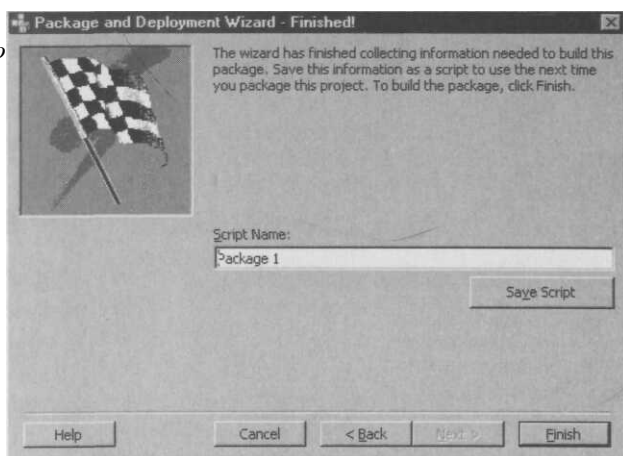
Figura 35.3
Il *Wizard recupera le informazioni necessarie per fornire i file di supporto per l'accesso a database.*

Se infine si fa clic su *Finish,* il Wizard preparerà Setup.Lst, comprimerà i file selezionati e copierà i file richiesti sul supporto di distribuzione o nella directory bersaglio. Se eseguite il programma d'installazione generato dal Package and Deployment Wizard, vedrete che il Wizard ha creato un programma d'installazione generico ma perfettamente rispettabile, come mostrato nella Figura 35.5. Notate che le parole in grigio nell'angolo in alto a sinistra della schermata di installazione sono, per default, il titolo del pacchetto e della distribuzione immessi nella finestra di dialogo *EXE Options* del progetto con la parola "Setup" aggiunta. Potete facilmente modificare questa visualizzazione di default modificando Setup.Lst, come vi mostrerò tra breve.

Figura 35.4 *L'ultimo pannello del Wizard vi permette di salvare un file di modello Setup Wizard (.Swt) basato sui dati che avete fornito al Wizard.*

File delle dipendenze

Un file delle dipendenze (.Dep) ha l'obiettivo di fornire informazioni sui requisiti runtime di un pacchetto e di una installazione, di un componente, o di un controllo. Notate che se la vostra applicazione comprende altri componenti o controlli, allora le dipendenze del vostro pacchetto comprendono le dipendenze di ogni altro oggetto incorporato.

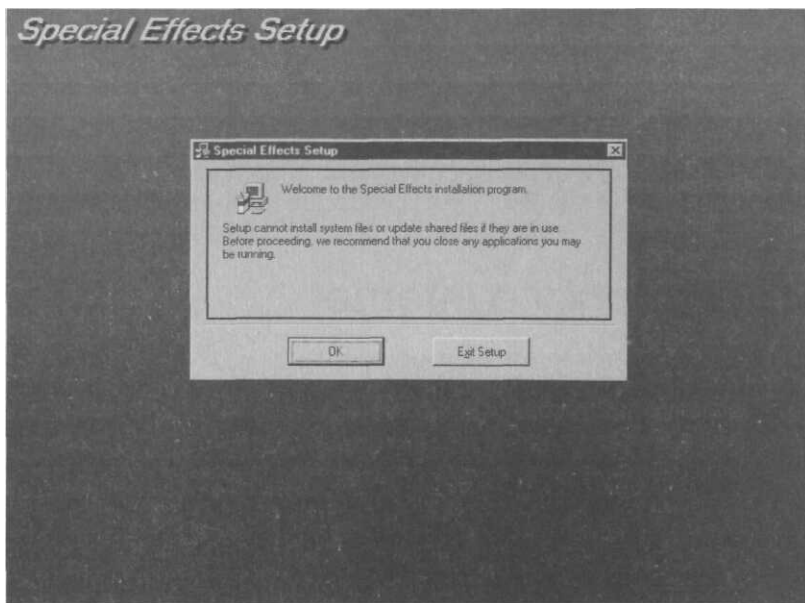
Potete usare Package and Deployment Wizard per l'unico scopo di creare un file .Dep, oppure potete scegliere di creare un file .Dep insieme con il vostro programma d'installazione. I file .Dep creati dal Wizard sono collocati nella stessa directory del componente o del progetto per cui sono stati creati.

I file .Dep possono essere creati per i componenti e per i progetti. Inoltre, le informazioni sulle dipendenze per lo stesso Visual Basic possono essere trovate nel file VB6dep.ini. Package and Deployment Wizard usa questo file per determinare le dipendenze per VB6.

File delle dipendenze per i componenti

I file delle dipendenze per i componenti sono usati per determinare le informazioni delle dipendenze per i controlli o per altri componenti. È bene fornire un file .Dep per ogni controllo o componente che si vuole distribuire. Un tipico file .Dep elenca tutti i file dipendenti, cioè i file richiesti perché i componenti o i controlli funzionino correttamente. Inoltre, il file .Dep contiene informazioni sulla versione e informazioni per il Registry. Per esempio, ecco i contenuti parziali di Comdlg32.Dep, il file delle dipendenze per il controllo finestra di dialogo comune:

```
[ComDlg32.ocx]
Dest=$(WinSysPath)
Register=$(DLLSelfRegister)
```

Figura 35.5 *Package and Deployment Wizard crea un programma d'installazione generico molto gradevole.*

```
Version=6.0.80.52
Uses1=ComCat.dll
Uses2=
CABFileName=ComDlg32.cab
CABDefaultURL=http:Ilactivex.microsoft.com/controls/vb5
CABINFFile=ComDlg32.inf
```

Il file VB6dep.ini

VB6dep.ini è usato per fornire *a* Setup Wizard un elenco di tutti i file richiesti da Visual Basic (dipendenze). VB6dep.ini comprende anche un elenco di tutti i riferimenti usati da VB e le informazioni che Setup Wizard può richiedere sul sistema su cui è installato VB (per esempio, la dimensione dei dischi rimovibili).

Nelle versioni di Visual Basic precedenti a VB5, il file che svolgeva lo scopo di VB6dep.ini era chiamato Swdepend.ini.

Quando Visual Basic viene installato su un sistema crea un file VB6dep.ini, che viene salvato nella directory PDWizard.

File delle dipendenze del progetto: assemblare il tutto

Quando Package and Deployment Wizard genera un programma di installazione per un progetto, mette le informazioni di dipendenza nel file Setup.Lst.

Notate che potete usare Setup Wizard solamente per creare un file delle dipendenze per un progetto, per i componenti o per un controllo.

Il Wizard analizza il progetto per scoprire le dipendenze e i riferimenti. Legge anche VB6dep.ini e tutti i file .Dep forniti dai componenti e dai controlli inclusi nel progetto. Le dipendenze combinate sono usate per generare l'elenco dei file e dei riferimenti richiesti per installare il pacchetto, cioè Setup.Lst.

Installazioni via Internet

Setup Wizard genera un programma d'installazione, comprendente il file del progetto compresso, e codice HTML di esempio che potete adattare alle vostre esigenze. (Troverete informazioni dettagliate sulla distribuzione Web nel Capitolo 27.) Se usate Setup Wizard per creare una installazione Web, assicuratevi di selezionare *Create Internet Download Setup* nel primo pannello.

È bene mettere tutti i file .Cab per l'utilizzo sul Web in una directory del vòstro sito Web, per una facile amministrazione.

La Figura 35.6 mostra la finestra di dialogo *Internet Distribution Location* del Wizard. Questa posizione potrebbe essere nella directory principale, oppure nella gerarchia del server Web, oppure in una qualsiasi altra posizione desideriate.

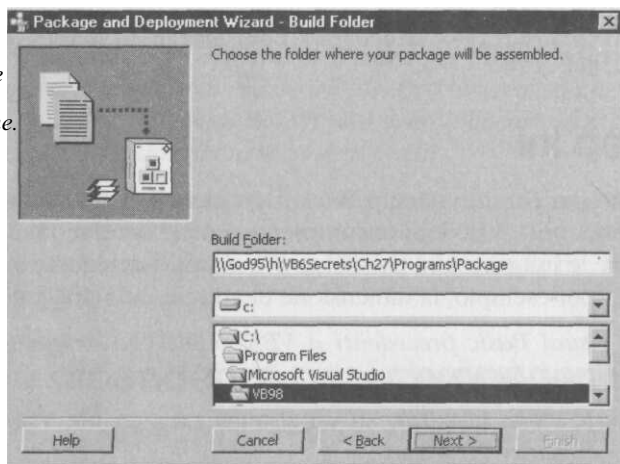
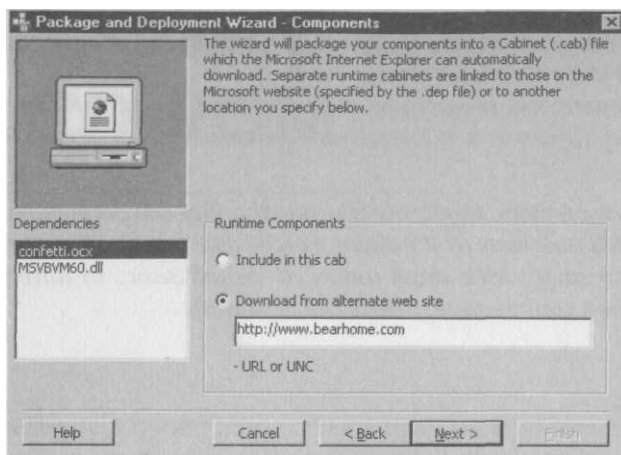
Figura 35.6
Dove specificare la destinazione per i file dell'installazione.

La Figura 35.7 mostra la finestra di dialogo *Internet Package,* che viene dopo la finestra di dialogo *Distribution Location.* Usate questa finestra per specificare se i file runtime di Visual Basic debbono essere scaricati dal sito Microsoft oppure da un sito alternativo. (Se il sistema bersaglio ha già i file runtime di Visual Basic, ovvero la Visual Basic Virtual Machine come viene spesso chiamata nel contesto Web, questi file non saranno scaricati.)

Figura 35.7 *La finestra di dialogo Internet Package vipermette di specificare un sito per scaricare i file di supporto (il default è da Microsoft).*

A meno che voi siate in una intranet senza accesso Web esterno, è preferibile scaricare i file di supporto da Microsoft (che è l'impostazione di default). In questo modo sarete sicuri di avere sempre l'ultima versione.

Modifica del progetto modello Setup1

Come ho già fatto notare nella descrizione di Package and Deployment Wizard di Visual Basic, il Wizard personalizza le installazioni per soddisfare le necessità degli specifici pacchetti modificando gli elementi in Setup.Lst che sono usati dal progetto Setupl.Vbp. Questo progetto si trova nella directory PDWizard\Setup1. Niente vi impedisce di introdurre le vostre modifiche, ma vi raccomando caldamente di fare una copia di backup di questo progetto prima di fare delle sperimentazioni.

In alcuni casi funzionerà bene creare una versione personalizzata di Setup1, per compilarla e perpermettere a Package and Deployment Wizard di generare automaticamente il vostro supporto di distribuzione. Certamente, se preferite, potete pianificare voi stessa tutta la distribuzione determinando i file necessari, comprìmendoli, copiando i file compressi nelle locazioni di distribuzione, e preparare un file Setup.Lst manualmente.

Se decidete di provare a modificare il progetto Setup1, sarete lieti di sapere che Microsoft fornisce dei commenti completi che vi guideranno. Il punto dove incominciare è l'evento `Form Load` per `frmSetup1`. Questa procedura contiene il codice che esegue l'installazione, principalmente chiamando le routine Setupl.Bas che operano sui dati forniti da Setup.Lst. Per esempio, l'istruzione

```
CopySection strINI_FILES
```

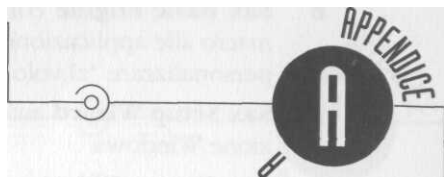
chiama la routine CopySection in Setupl.Bas che prova a copiare tutti i file elencati in una sezione di Setup.Lst. Potete facilmente aggiungere all'evento Form Load una finestra di dialogo che richieda all'utente dei dati.
Per modificare l'aspetto del programma d'installazione potete cambiare l'aspetto di frmSetupl. Potete applicare a questo form qualsiasi tecnica grafica descritta nel Capitolo 19.

Lo stesso Setupl.Bas contiene numerose routine di utility che permettono la manipolazione dei file e dell'ambiente di Windows. Poiché queste procedure sono distribuite liberamente e sono state scritte dagli autori di Visual Basic, in altre parole dalla Microsoft, e sono ben commentate, è bene che le studiate.

Riepilogo

I programmi d'installazione sono importanti perché forniscono agli utenti la prima impressione del vostro pacchetto e perché una buona procedura d'installazione può assicurare che il programma funzioni correttamente. Package and Deployment Wizard di Visual Basic è facile da usare e fornisce un metodo potente per generare le procedure d'installazione. Package and Deployment Wizard usa un file di avvio per eseguire un progetto di Visual Basic compilato che gestisce il lavoro di impostazione di un pacchetto e di una distribuzione. Potete modificare il codice sorgente contenuto in questo progetto per personalizzare l'aspetto e il funzionamento del programma d'installazione.

- Avete imparato come usare le informazioni ottenute da Package and Deployment Wizard.
- Avete imparato quello che fa il Wizard.
- Ho trattato i concetti dell'avvio.
- Ho trattato i file delle dipendenze (.Dep).
- Avete imparato come creare le installazioni via Internet.
- Ho spiegato il ruolo della rimozione automatica.
- Vi ho insegnato come usare Package and Deployment Wizard.
- Vi ho mostrato come personalizzare Setupl.Exe.

CHE COSA C'È NEL CD-ROM

Il CD-ROM allegato a questo libro contiene tutti i progetti sorgente in Visual Basic sviluppati e commentati nel testo, nonché moduli di codice di utility. I progetti e i moduli in codice sorgente sono organizzati per capitolo. Così per esempio, per trovare il codice sorgente del Capitolo 15, dovete cercarlo nella directory Source-Code\Chl5 sul CD-ROM.

Ho chiesto a vari fra i principali sviluppatori di prodotti per Visual Basic di fornire materiale utile per il CD-ROM. Ciascun prodotto ha avuto la propria directory, con un nome che identifica il produttore. Per esempio, troverete l'offerta di InstallShield nella directory 3-Pty\Ishield sul CD.

Normalmente, prodotti diversi dello steso produttore hanno la propria sottodirectory. Ogni produttore ha fornito i propri programmi di installazione, che dovrete eseguire per installare il prodotto sul vostro sistema. Ecco in breve i prodotti di terze parti che troverete sul CD-ROM:

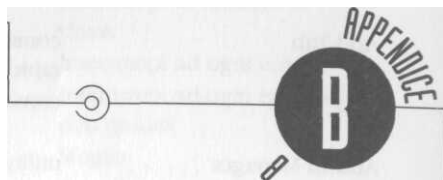
- **Dalla Blue Sky Software Corporation:** la Blue Sky è il produttore principale di strumenti per la creazione di Guide in linea. Sul CD-ROM, troverete versioni trial per 30 giorni di RoboHELP HTML, un programma che aiuta a creare sistemi di guida basati su HTML, e What's This? Help Composer, strumento da usare per generare funzionalità di Guida per descrizioni rapide.
- **Dalla Desaware:** troverete versioni di valutazione degli strumenti ActiveX della Desaware per gli sviluppatorim in particolare:
  - **SpyWorks**
  - **Storage Tools**
  - **Version Tools**
  - **Il nuovo ActiveX Gallimaufry**

  I prodotti della Desaware hanno un obiettivo comune: aiutare chi programma in Visual Basic a diventare un grande programmatore, a sfruttare a pieno tutte le capacità di Visual Basic e ad andare anche oltre quelle capacità, quando necessario.
- **Dalla InstallShield:** la InstallShield Software Corporation, il maggior pro-

duttore mondiale di software per la creazione di programmi personalizzati di installazione, ci ha fornito una edizione di valutazione di InstallShield Express versione 2.02.

- **Dalla Sax Software:** la Sax Software ci ha fornito edizioni di valutazione dei prodotti seguenti:
  - **Sax Basic Engine** consente di aggiungere facilmente un linguaggio macro alle applicazioni Visual Basic, in modo che gli utenti le possano personalizzare "al volo".
  - **Sax Setup Wizard** automatizza la creazione di programmi di installazione Windows.
  - **Sax Comm Objects** consente di aggiungere facilmente funzioni di comunicazione seriale alle applicazioni Visual Basic.
  - **Sax mPower 98** consente di costruire facilmente applicazioni di workflow potenti e collaborative, con connettività illimitata a database.
- **Dalla VideoSoft:** nel CD-ROM si possono trovare edizioni di valutazione dei programmi seguenti:
  - **VS-Ocx 6.0**
  - **VSView 3.0**
  - **VSFlex 3.0**
  - **VSReports**
  - **VSData**
  - **VSDirect**
  - **VSSpell**
  - **VSDocX**

  Questi prodotti, aggiunte importanti al repertorio di qualsiasi programmatore, sono pienamente funzionali, ma visualizzano un banner, quando viene caricato un progetto che li contiene, in cui si specifica che è in uso una versione di valutazione. Gli strumenti della VideoSoft sono fra i pia importanti e racchiudono una gamma di funzionalità enorme. Tutti i programmi dovrebbero averli.

CORRISPONDENZE INGLESE-ITALIANO

Nel testo ci siamo attenuti alla versione inglese di Visual Basic 6.0, dato che molti programmatori anche in Italia usano questa versione e inoltre è indispensabile conoscerla per gli esami di certificazione Microsoft. In questa Appendice forniamo una doppia tabella di consultazione per chi ha a disposizione la versione italiana: nella prima tabella si troveranno in ordine alfabetico le espressioni inglesi utilizzate nel libro, relative all'interfaccia di Visual Basic 6, con i corrispondenti italiani; nella seconda si troveranno le voci italiane in ordine alfabetico, con i corrispondenti inglesi. In questo modo si può risalire dal testo inglese al corrispondente italiano (ove esista, beninteso: non tutto è tradotto) e dall'interfaccia italiana all'espressione inglese, per orientarsi nel libro.

Inglese	**Situazione di occorrenza**	**Italiano**
ActiveX Component	opzione della scheda Component della finestra di dialogo Project Properties (proprietà progetto)	Componente ActiveX
ActiveX Control Interface Wizard	aggiunta	Creazione guidata interfaccia controlli ActiveX
ActiveX Control Test Container	utility	ActiveX Control Test Container
ActiveX Designers	comando	Finestre di progettazione
ActiveX Document Migration Wizard	aggiunta di VB6	Conversione guidata documenti ActiveX
Add	comando	Inserisci
Add Class Module	comando	Inserisci modulo di classe
Add Custom Member	finestra di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Aggiungi membro personalizzato
Add Form	comando	Inserisci form
Add Form	finestra di dialogo	Inserisci form
Add Form	pulsante	Inserisci form
Add Module	pulsante	Inserisci modulo

Inglese	Situazione di occorrenza	Italiano
Add Module	cornando	Inserisci modulo
Add Procedure	comando	Inserisci routine
Add Property Page	comando del menu Project	Inserisci pagine proprietà
Add Step	pulsante di Wizard Manager	Aggiunge un nuovo passaggio alla creazione guidata
Add Tab	comando di menu di scelta rapida	Aggiungi scheda
Add Watch	comando di debug	Aggiungi espressione di controllo
Add-In Manager	utility	Gestione aggiunte
Add-In Toolbar	aggiunta di VB6	Barra degli strumenti Aggiunte
Add-Ins	menu	Aggiunte
Advanced	scheda	Avanzate
Advanced Optimizations	pulsante della scheda Compile (Compila) nella finestra di dialogo Project Properties (Proprietà progetto)	Ottimizzazioni avanzate
Aliasing	opzione della finestra di dialogo Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Non prevedere aliasing
Align	comando	Allinea
Allow Unrounded Floating Point Operations	opzione della finestra di dialogo Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Operazioni con virgola mobile senza arrotondamento
Alphabetic	scheda in Properties	Alfabetico
Apartment-threaded	opzione della scheda General (Generale) della finestra di dialogo Project Properties (Proprietà progetto)	Con Aprtment-threading
API Text Viewer	utility	API Text Viewer
Auto Data Tips	opzione	Descrizione dati automatica
Auto List Members	opzione	Elenco membri automatico
Auto Quick Info	opzione	Informazioni rapide automatiche
Auto Syntax Check	opzione	Controllo automatico sintassi
AVIEditor	utility	Editor di file AVI
Back	pulsante	Indietro
Background Compile	opzione	Compila in background

Inglese	Situazione di occorrenza	Italiano
Binary Compatibility	opzione della finestra di dialogo Project Properties (Proprietà progetto)	Compatibilita binaria
Bookmarks	comando	Segnalibri
Break in Class Module	opzione cattura errori	Interrompi in modulo di classe
Break on All Errors	opzione cattura errori	Interrompi ad ogni errore
Break on Unhandled Errors	opzione cattura errori	Interrompi ad ogni errore non gestito
Browse	pulsante	Sfoglia
Cancel	pulsante	Annulla
Caption	cas testo	Caption
Categorized	scheda in Properties (Proprietà)	Per categoria
Center in Form	comando	Centra nel form
Class Builder	aggiunta	Creazione guidata classi
Classes	pannello	Classi
Clear All Breakpoints	strum di debug	Rimuovi punti di interruzione
Clipboard Object Constants	argomento della Guida in linea	Costanti oggetto Clipboard
Code	finestra	Codice
Code Settings	riquadro	Impostazioni codice
Collapse Proj. Hides Windows	opzione	Comprimi progetto e nascondi finestre
Command	pulsante	Comando
Compile	scheda di Project Properties (Proprietà progetto)	Compila
Compile On Demand	opzione	Compila su richiesta
Complete Word	ausilio programmazione	Completa parola
Component	scheda della finestra di dialogo Project Properties (Proprietà progetto)	Componente
Components	comando	Componenti
Components	finestra di dialogo	Componenti
Connect Property Page	finestra di dialogo	Collega pagine proprietà
Copy	comando	Copia
Create Custom Interface Member	finestra di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Crea membri personalizzati
Create Default Project	opzione	Crea nuovo progetto
Create Embedded Object	comando del menu di scelta rapida di un controllo OLE	Crea oggetto incorporato
Create From File	pulsante della finestra di dialogo Insert Object (Inserisci oggetto)	Crea dal file

Inglese	Situazione di occorrenza	Italiano
Create Link	comando del menu di scelta rapida di un controllo OLE	Crea collegamento
Create Symbolic Debug Info	opzione della scheda Compile (Compila) nella finestra di dialogo Project Properties (Proprietà progetto)	Crea informazioni codificate di debug
Cut	comando	Taglia
Data Object Viewer	utility	Visualizzatore DataObject
DDE Spy	utility	DDE Spy
Debug	menu	Debug
Delete	comando	Elimina
Delete Embedded Object	comando del menu di scelta rapida di un controllo OLE	Elimina oggetto incorporato
Delete Linked Object	comando del menu di scelta rapida di un controllo OLE	Elimina oggetto collegato
Depens	utility	Dipende
Designers	scheda	Finestre di progettazione
Details	pannello	Dettagli
DocFile Viewer	utility	Visualizzatore DocFile
Docking	scheda	Ancoraggio
Don't Save Changes	opzione	Non salvare
Don't Show in Property Browser	casella della finestra di dialogo Procedure Attributes (Attributiroutine)	Non visualizzare nella finestra Proprietà
Edit	menu	Modifica
Edit Watch	finestra di dialogo	Modifica espressione di controllo
Editor	scheda	Editor
Editor Format	scheda	Formato editor
Environment	scheda	Ambiente•
Error Lookup	utility	Ricerca errori
Exit	comando	Esci
Favor Pentium pro	opzione della scheda Compile (Compila) nella finestra di dialogo Project Properties (Proprietà progetto)	Ottimizza per Pentium Pro
File	menu	File
Files of Type	cas riep	Tipo file
Find	comando	Trova
Find Next	comando	Trova successivo
Font	finestra	Carattere
Form	comando	Form
Form Layout	finestra	Disposizione form
Format	menu	Formato
Forms	opzione	Form

Inglese	Situazione di occorrenza	Italiano
General	scheda	Generale
Help	menu	Guida
Hide This Member	casella della finestra di dialogo Procedure Attributes (Attributi routine)	Nascondi questo membro
Horizontal Spacing	comando	Spaziatura orizzontale
Immediate	finestra	Immediata
Indent	comando	Aumenta rientro
Insert Button	pulsante	
Insert File	comando	Inserisci file
Insert Object	Finestra di dialogo	Inserici oggetto
Insert Step	pulsante di Wizard Manager	Inserisce un passaggio immediatamente dopo il passaggio corrente
Insert Tab	pulsante	Inserisci scheda
Insertable Objects	scheda	Oggetti inseribili
Link	casella della finestra di dialogo Insert Object (Inserisci oggetto)	Collegamento
List Constants	comando	Elenca costanti
List Properties/Methods	comando	Elenca proprietà/metodi
Load Behavior	riquadro della finestra Add-In Manager (Gestione aggiunte)	Caricamento
Loaded/Unloaded	opzione della finestra Add-In Manager (Gestione aggiunte)	Caricato/scaricato
Lock Controls	comando	Blocca i controlli
Make	comando	Crea
Make Project	finestra di dialogo	Crea progetto
Make Same Size	comando	Rendi della stessa dimensione
Margin Indicator Bar	opzione della scheda Editor Format (Formato editor) della finestra di dialogo Options (Opzioni)	Barra indicatori
MDI Form	comando	Form MDI
Members	pannello	Membri
More ActiveX Designers	comando	
Move Step Down One	pulsante di Wizard Manager	Sposta il passaggio corrente verso il basso
Move Step Up One	pulsante di Wizard Manager	Sposta il passaggio corrente verso l'alto
New	comando	Nuovo
New Project	comando del menu File	Nuovo progetto
New Tab Name	finestra di dialogo	Nuovo nome scheda
Next	pulsante	Successivo

Inglese	Situazione di occorrenza	Italiano
Notify When Changing Shared Project Items	opzione	Notifica modifiche a elementi condivisi del progetto
Object	casella di riepilogo	Oggetto
Object Browser	comando	Visualizzatore oggetti
Open	pulsante	Apri
Open Project	comando del menu File	Apri progetto
Options	comando	Opzioni
Options	Finestra di dialogo	Opzioni
Order	comando	Ordinamento
Outdent	comando	Riducirientro
Package and Deployment Wizard	aggiunta di VB6	Creazione guidata pacchetti di installazione
Parameter Info	comando	Informazioni parametri
Paste	comando	Incolla
Paste Special	comando del menu di scelta rapida di un controllo OLE	Incolla speciale
Procedure	casella di riepilogo	Routine
Process Viewer	utility	Visualizzatore processo
Project	menu	Progetto
Project	finestra	Progetto
Project Descriptiont	nella finestra di dialogo Project Properties (Proprietà progetto)	Descrizione progetto
Project Explorer	comando	Gestione progetti
ProjectName	nella finestra di dialogo Project Properties (Proprietà progetto)	Nome progetto
Project Options	finestra di dialogo	Opzioni progetto
Project Properties	finestra di dialogo	Proprietà progetto
Prompt for Project	opzione	Seleziona progetto
Prompt To Save Changes,	opzione	Salva con conferma
Properties	finestra	Proprietà
Property is data bound	opzione della finestra di dialogo Procedure Attributes (Attributi routine)	Proprietà associata a dati
Property Pages	finestra	Pagine proprietà
Quick Info	comando	Informazionirapide
Quick Watch	strum di debug	Controllo immediato
Redo	comando	Ripeti
References	finestra di dialogo	Riferimenti
Refresh Step List	pulsante di Wizard Manager	Aggiorna l'elenco dei passaggi

Inglese	Situazione di occorrenza	Italiano
Remove Array Bound Checks	opzione della finestra di dialogo Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Rimuovi codice di verifica degli indici delle matrici
Remove Floating Point Error Checks	opzione della finestra di dialogo Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Rimuovi controllo degli errori di virgola mobile
Remove Information About Unused ActiveX Controls	nella scheda Make (Crea) di Project Properties (Proprietà progetto)	Rimuovi informazioni sui controlli ActiveX non utilizzati
Remove Integer Overflow Checks	opzione della finestra di dialogo Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Rimuovi controllo dell'overflow di interi
Remove Sarfe Pentium FDIV Checks	opzione della finestra di dialogo Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Rimuovi controlli di sicurezza su FDIV Pentium
Replace	comando	Sostituisci
Require License Key	opzione nella scheda General (Generale) della finestra di dialogo Project Properties (Proprietà progetto)	Richiedi codice licenza
Require Variable Declaration.	opzione della finestra di dialogo Options (Opzioni)	Dichiarazione di variabili obbligatoria
ROT Viewer	utility	Visualizzatore ROT
Run	menu	Esegui
Run to Cursor	comando	Esegui fino al cursore
Save	pulsante	Salva
Save	comando	Salva
Save Changes	opzione	Salva senza conferma
Save Project	comando del menu File	Salva progetto
Save Project As	comando del menu File	Salva progetto con nome
Save Project Group	comando del menu File	Salva gruppo di progetti
Save Project Group As	comando del menu File	Salva gruppo di progetti con nome
SDI Development Environment	casella di opzione	Ambiente di sviluppo SDI

Inglese	Situazione di occorrenza	Italiano
Select Interface Members	finestra di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Seleziona membri interfaccia
Selected Names	casella di riepilogo di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Nomi selezionati
Set Attributes	finestra di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Imposta attributi
Set Mapping	finestra di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Imposta associazioni
Set Next Statement	comando del menu Debug	Imposta istruzione successiva
Show in DataBindings Collection at design time	opzione della finestra di dialogo Procedure Attributes (Attributi routine)	Mostra nell'insieme DataBindings in fase di progettazione
Show Next Statement	comando del menu Debug	Mostra istruzione successiva
Show Templates For	riquadro	Mostra modelli per
Show ToolTips	opzione	Mostra descrizione comandi
Single-threaded	opzione della scheda General (Generale) della finestra di dialogo Project Properties (proprietà progetto)	A thread singolo
Size to Grid	comando	Dimensiona alla griglia
Source Code Control	aggiunta di VB6	Controllo del codice sorgente
Start Model	Riquadro della scheda Component (Componente) della finestra di dialogo Project Properties (Proprietà progetto)	Modello di avvio
Start With Full Compile	comando dal menu Run (Esegui)	Avvia con compilazione completa
Startup Object	opzione della scheda General (Generale) della finestra di dialogo Project Properties (proprietà progetto)	Oggetto di avvio
Step Into	comando del menu Debug	Esegui istruzione
Step Out	comando del menu Debug	Esci da istruzione/routine
Step Over	comando del menu Debug	Esegui istruzione/routine
Thread per Object	opzione della scheda General (Generale) della finestra di dialogo Project Properties (proprietà progetto)	Thread per oggetto

Inglese	Situazione di occorrenza	Italiano
Thread Pool	opzione della scheda General (Generale) della finestra di dialogo Project Properties (proprietà progetto)	Pool di thread
Threading Model	opzione della scheda General (Generale) della finestra di dialogo Project Properties (proprietà progetto)	Modello di threading
Toggle Breakpoints	comando del menu Debug	Imposta/rimuovi punto di interruzione
Toolbars	Comando del menu View (Visualizza)	Barre degli strumenti
Tools	menu	Strumenti
Undo	comando	Annulla
VB T-SQL Debugger	aggiunta di VB6	Debugger VB T-SQL
VB6 API Viewer	aggiunta di VB6	Visualizzatore API VB6
VB6 Application Wizard	aggiunta di VB6	Creazione guidata applicazioni VB6
VB6 Class Builder Utility	aggiunta di VB6	Creazione guidata classi VB6
VB6 Data Form Wizard	aggiunta di VB6	Creazione guidata form dati
VB6 Wizard Manager	aggiunta di VB6	Creazione operazioni guidate VB6
Version	scheda	Versione
Vertical Spacing	comando	Spaziatura verticale
View	menu	Visualizza
View Code	pulsante del menu scelta rapida	Visualizza codice
Watch	finestra	Espressioni di controllo
When a Program Starts	opzione	All'avvio di un programma
Window	menu	Finestra
ZoomIn	utility	Zoom avanti

Italiano	Situazione di occorrenza	Inglese
A thread singolo	opzione della scheda General (Generale) della finestra di dialogo Project Properties (Proprietà progetto)	Single-threaded
ActiveX Control Test Container	utility	ActiveX Control Test Container
Aggiorna l'elenco dei passaggi	pulsante di Wizard Manager	Refresh Step List
Aggiunge un nuovo passaggio alla creazione guidata	pulsante di Wizard Manager	Add Step
Aggiungi espressione di controllo	strum di debug	Add Watch
Aggiungi membro personalizzato	finestra di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Add Custom Member
Aggiungi scheda	comando di menu di scelta rapida	Add Tab
Aggiunte	menu	Add-Ins
Alfabetico	scheda in Properties	Alphabetic
All'avvio di un programma	opzione	When a Program Starts
Allinea	comando	Align
Ambiente	scheda	Environment
Ambiente di sviluppo SDI	casella di opzione	SDI Development Environment
Ancoraggio	scheda	Docking
Annulla	pulsante	Cancel
Annulla	comando	Undo
API Text Viewer	utility	API Text Viewer
Apri	pulsante	Open
Apri progetto	comando del menu File	Open Project
Aumenta rientro	comando	Indent
Avanzate	scheda	Advanced
Avvia con compilazione completa	comando dal menu Run (Esegui)	Start With Full Compile
Barra degli strumenti Aggiunte	aggiunta di VB6	Add-In Toolbar
Barra indicatori	opzione della scheda Editor Format (Formato editor) della finestra di dialogo Options (Opzioni)	Margin Indicator Bar
Barre degli strumenti	comando del menu View (Visualizza)	Toolbars
Blocca i controlli	comando	Lock Controls
Caption	cas testo	Caption
Carattere	finestra	Font

Italiano	Situazione di occorrenza	Inglese
Caricamento	riquadro della finestra Add-In Manager (Gestione aggiunte)	Load Behavior
Caricato/scaricato	opzione della finestra Add-In Manager (Gestione aggiunte)	Loaded/Unloaded
Centra nel form	comando	Center in Form
Classi	pannello	Classes
Codice	finestra	Code
Collega pagine proprietà	finestra di dialogo	Connect Property Page
Collegamento	casella della finestra di dialogo Inserì Object (Inserisci oggetto)	Link
Comando	pulsante	Command
Compatibilitabinaria	opzione della finestra di dialogo Project Properties (Proprietà progetto)	Binary Compatibility
Compila	scheda di Project Properties (Proprietà progetto)	Compile
Compila in background	opzione	Background Compile
Compila su richiesta	opzione	Compile On Demand
Completa parola	ausilio programmazione	Complete Word
Componente	scheda della finestra di dialogo Project Properties (Proprietà progetto)	Component
Componente ActiveX	opzione della scheda Component della finestra di dialogo Project Properties (proprietà progetto)	ActiveX Component
Componenti	comando	Components
Componenti	finestra di dialogo	Components
Comprimi progetto e nascondi finestre	opzione	Collapse Proj. Hides Windows
Con Aprtment-threading	opzione della scheda General (Generale) della finestra di dialogo Project Properties (Proprietà progetto)	Apartment-threaded
Controllo automatico sintassi	opzione	Auto Syntax Check
Controllo del codice sorgente	aggiunta di VB6	Source Code Control
Controllo immediato	comando di debug	Quick Watch
Conversione guidata documenti ActiveX	aggiunta di VB6	ActiveX Document Migration Wizard
Copia	comando	Copy
Costanti oggetto Clipboard	argomento della Guida in linea	Clipboard Object Constants
Crea	comando	Make
Crea collegamento	comando del menu di scelta rapida di un controllo OLE	Create Link

Italiano	Situazione di occorrenza	Inglese
Crea dal file	pulsante della finestra di dialogo Insert Object (Inserisci oggetto)	Create From File
Crea informazioni codificate di debug	opzione della scheda Compile (Compila) nella finestra di dialogo Project Properties (Proprietà progetto)	Create Symbolic Debug Info
Crea membri personalizzati	finestra di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Create Custom Interface Member
Crea nuovo progetto	opzione	Create Default Project
Crea oggetto incorporato	comando del menu di scelta rapida di un controllo OLE	Create Embedded Object
Crea progetto	finestra di dialogo	Make Project
Creazione guidata applicazioni VB6	aggiunta di VB6	VB6 Application Wizard
Creazione guidata classi	aggiunta	Class Builder
Creazione guidata classi VB6	aggiunta di VB6	VB6 Class Builder Utility
Creazione guidata form dati	aggiunta di VB6	VB6 Data Form Wizard
Creazione guidata interfaccia controlli ActiveX	aggiunta	ActiveX Control Interface Wizard
Creazione guidata pacchetti di installazione	aggiunta di VB6	Package and Deployment Wizard
Creazione operazioni guidate VB6	aggiunta di VB6	VB6 Wizard Manager
DDE Spy	utility	DDE Spy
Debug	menu	Debug
Debugger VB T-SQL	aggiunta di VB6	VB T-SQL Debugger
Descrizione dati automatica	opzione	Auto Data Tips
Descrizione progetto	nella finestra di dialogo Project Properties (Proprietà progetto)	Project Descriptiont
Dettagli	pannello	Details
Dichiarazione di variabili obbligatoria	opzione della finestra di dialogo Options (Opzioni)	Require Variable Declaration.
Dimensiona alla griglia	comando	Size to Grid
Dipende	utility	Depens
Disposizione form	finestra	Form Layout
Editor	scheda	Editor
Editar di file AVI	utility	AVI Editor
Elenca costanti	comando	List Constants
Elenca proprietà/metodi	comando	List Properties/Methods
Elenco membri automatico	opzione	Auto List Members
Elimina	comando	Delete

Italiano	Situazione di occorrenza	Inglese
Elimina oggetto collegato	comando del menu di scelta rapida di un controllo OLE	Delete Linked Object
Elimina oggetto incorporato	comando del menu di scelta rapida di un controllo OLE	Delete Embedded Object
Esci	comando	Exit
Esci da istruzione/routine	comando del menu Debug	Step Out
Esegui	menu	Run
Esegui fino al cursore	comando	Run to Cursor
Esegui istruzione	comando del menu Debug	Step Into
Esegui istruzione/routine	comando del menu Debug	Step Over
Espressioni di controllo	finestra	Watch
File	menu	File
Finestra	menu	Window
Finestre di progettazione	comando	ActiveX Designers
Finestre di progettazione	scheda	Designers
Form	comando	Form
Form	opzione	Forms
Form MDI	comando	MDI Form
Formato	menu	Format
Formato editor	scheda	Editor Format
Generale	scheda	General
Gestione aggiunte	utility	Add-In Manager
Gestione progetti	comando	Project Explorer
Guida	menu	Help
Immediata	finestra	Immediate
Imposta associazioni	finestra di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Set Mapping
Imposta attributi	finestra di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Set Attributes
Imposta istruzione successiva	comando del menu Debug	Set Next Statement
Imposta/rimuovi punto di interruzione	comando del menu Debug	Toggle Breakpoints
Impostazioni codice	riquadro	Code Settings
Incolla	comando	Paste
Incolla speciale	comando del menu di scelta rapida di un controllo OLE	Paste Special
Indietro	pulsante	Back
Informazioni parametri	comando	Parameter Info
Informazioni rapide	comando	Quick Info
Informazioni rapide automatiche	opzione	Auto Quick Info

Italiano	Situazione di occorrenza	Inglese
Inserici oggetto	finestra di dialogo	Insert Object
Inserisce un passaggio immediatamente dopo il passaggio corrente	pulsante di Wizard Manager	Insert Step
Inserisci	comando	Add
Inserisci file	comando	Insert File
Inserisci form	comando	Add Form
Inserisci form	finestra di dialogo	Add Form
Inserisci form	pulsante	Add Form
Inserisci modulo	pulsante	Add Module
Inserisci modulo	comando	Add Module
Inserisci modulo di classe	comando	Add Class Module
Inserisci pagine proprietà	comando del menu Project	Add Property Page
Inserisci routine	comando	Add Procedure
Inserisci scheda	pulsante	Insert Tab
Interrompi ad ogni errore	opzione cattura errori	Break on All Errors
Interrompi ad ogni errore non gestito	opzione cattura errori	Break on Unhandled Errors
Interrompi in modulo di classe	opzione cattura errori	Break in Class Module
Membri	pannello	Members
Modello di avvio	riquadro della scheda Component (Componente) della finestra di dialogo Project Properties (Proprietà progetto)	Start Model
Modello di threading	opzione della scheda General (Generale) della finestra di dialogo Project Properties (proprietà progetto)	Threading Model
Modifica	menu	Edit
Modifica espressione di controllo	finestra di dialogo	Edit Watch
Mostra descrizione comandi	opzione	Show ToolTips
Mostra istruzione successiva	comando del menu Debug	Show Next Statement
Mostra modelli per	riquadro	Show Templates For
Mostra nell'insieme DataBindings in fase di progettazione	opzione della finestra di dialogo Procedure Attributes (Attributi routine)	Show in DataBindings Collection at design time
Nascondi questo membro	casella della finestra di dia-	Hide This Member
Nome progetto	nella finestra di dialogo Project Properties (Proprietà progetto)	Project Name

Italiano	Situazione di occorrenza	Inglese
Nomi selezionati	casella di riepilogo di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Selected Names
Non prevedere aliasing	opzione della finestra di dialogo Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Aliasing
Non salvare	opzione	Donit Save Changes
Non visualizzare nella finestra Proprietà	casella della finestra di dialogo Procedure Attributes (Attributi routine)	Don't Show in Property Browser
Notifica modifiche a elementi condivisi del progetto	opzione	Notify When Changing Shared Project Items
Nuovo	comando	New
Nuovo nome scheda	finestra di dialogo	New Tab Name
Nuovo progetto	comando del menu File	New Project
Oggetti inseribili	scheda	Insertable Objects
Oggetto	casella di riepilogo	Object
Oggetto di avvio	opzione della scheda General (Generale) della finestra di dialogo Project Properties (proprietà progetto)	Startup Object
Operazioni con virgola mobile senza arrotondamento	opzione della finestra di dialogo Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Allow Unrounded Floating Point Operations
Opzioni	comando	Options
Opzioni	finestra di dialogo	Options
Opzioni progetto	finestra di dialogo	Project Options
Ordinamento	comando	Order
Ottimizza per Pentium Pro	opzione della scheda Compile (Compila) nella finestra di dialogo Project Properties (Proprietà progetto)	Favor Pentium pro
Ottimizzazioni avanzate	pulsante della scheda Compile (Compila) nella finestra di dialogo Project Properties (Proprietà progetto)	Advanced Optimizations
Pagine proprietà	finestra	Property Pages
Per categoria	scheda in Properties (Proprietà)	Categorized

Italiano	Situazione di occorrenza	Inglese
Pooldithread	opzione della scheda General (Generale) della finestra di dialogo Project Properties (proprietà progetto)	Thread Pool
Progetto	menu	Project
Progetto	finestra	Project
Proprietà	finestra	Properties
Proprietà associata a dati	opzione della finestra di dialogo Procedure Attributes (Attributi routine)	Property is data bound
Proprietà progetto	finestra di dialogo	Project Properties
Rendidellastessadimensione	comando	Make Same Size
Ricerca errori	utility	Error Lookup
Richiedi codice licenza	opzione nella scheda General (Generale) della finestra di dialogo Project Properties (Proprietà progetto)	Require License Key
Riduci rientro	comando	Outdent
Riferimenti	finestra di dialogo	References
Rimuovi codice di verifica degli indici delle matrici	opzione della finestra di dial°g° Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Remove Array Bound Checks
Rimuovi controlli di sicurezza su FDIV Pentium	opzione della finestra di dialog° Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Remove Sarfe Pentium FDIV Checks
Rimuovi controllo degli errori di virgola mobile	opzione della finestra di dialogo Advanced Optimizations (Ottimizzazioni *avanzate), dalla* finestra di dialogo Project Properties (Proprietà progetto)	Remove Floating Point Error Checks
Rimuovi controllo dell'overflow di interi	opzione della finestra di dialogo Advanced Optimizations (Ottimizzazioni avanzate), dalla finestra di dialogo Project Properties (Proprietà progetto)	Remove Integer Overflow Checks
Rimuovi informazioni sui controlli ActiveX non utilizzati	nella scheda Make (Crea) di Project Properties (Proprietà progetto)	Remove Information About Unused ActiveX Controls

Italiano	Situazione di occorrenza	Inglese
Rimuovi punti di interruzione	strum di debug	Clear All Breakpoints
Ripeti	comando	Redo
Routine	casella di riepilogo	Procedure
Salva	pulsante	Save
Salva	comando	Save
Salva con conferma	opzione	Prompt To Save Changes,
Salva gruppo di progetti	comando del menu File	Save Project Group
Salva gruppo di progetti con nome	comando del menu File	Save Project Group As
Salva progetto	comando del menu File	Save Project
Salva progetto con nome	comando del menu File	Save Project As
Salva senza conferma	opzione	Save Changes
Segnalibri	comando	Bookmarks
Seleziona membri interfaccia	finestra di ActiveX Control Interface Wizard (Creazione guidata interfaccia controlli ActiveX)	Select Interface Members
Seleziona progetto	opzione	Prompt for Project
Sfoglia	pulsante	Browse
Sostituisci	comando	Replace
Spaziatura orizzontale	comando	Horizontal Spacing
Spaziatura verticale	comando	Vertical Spacing
Sposta il passaggio corrente verso il basso	pulsante di Wizard Manager	Move Step Down One
Sposta il passaggio corrente verso l'alto	pulsante di Wizard Manager	Move Step Up One
Strumenti	menu	Tools
Successivo	pulsante	Next
Taglia	comando	Cut
Thread per oggetto	opzione della scheda General (Generale) della finestra di dialogo Project Properties (proprietà progetto)	Thread per Object
Tipo file	cas riep	Files of Type
Trova	comando	Find
Trova successivo	comando	Find Next
Versione	scheda	Version
Visualizza	menu	View
Visualizza codice	pulsante del menu scelta rapida	View Code
Visualizzatore API VB6	aggiunta di VB6	VB6 API Viewer
Visualizzatore DataObject	utility	Data Object Viewer
Visualizzatore DocFile	utility	DocFile Viewer
Visualizzatore oggetti	comando	Object Browser
Visualizzatore processo	utility	Process Viewer

Italiano	Situazione di occorrenza	Inglese
VisualizzatoreROT	utility	ROT Viewer
Zoomavanti	utility	ZoomIn
	pulsante	Insert Button
	comando	More ActiveX Designers

INDICE ANALITICO

D

E

F

N

O

P

Q

R

ISBN	Titolo	Prezzo
88-7303-450-0	I segreti di Windows 98	L 98.000
88-7303-437-3	Windows 98 Guida completa	L. 69.000
88-7303-447-0	Windows 98 For Dummies	L. 36.000
88-7303-459-4	Windows 98 For Dummies Espresso	L. 19.000
88-7303-436-5	Windows 98 flash	L. 16.000
88-7303-461-6	Windows 98 Tutto&Oltre	L. 88.000
88-7303-451-9	Windows 98 Installazione e configurazione	L. 88.000
88-7303-438-1	I segreti della programmazione in Windows 98	L. 98.000
88-7303-460-8	L'API di Windows 98 For Dummies Espresso	L. 19.000
88-7303-398-9	Windows 98 Anteprima	L. 19.000
88-7303-479-9	Programmare in Windows 98/NT Tutto&Oltre	L. 88.000
88-7303-471-3	Introduzione a Windows NT 5	L. 45.000
88-7303-346-6	Windows 95 For Dummies Espresso	L. 19.000
88-7303-377-6	Windows 95 flash Seconda edizione	L. 16.000
88-7303-427-6	Windows 95 per tutti	L. 32.000
88-7303-298-2	I segreti di Windows NT Server 4	L. 78.000
88-7303-141-2	I segreti di Windows 95	L. 92.500
88-7303-340-7	Windows NT For Dummies	L. 32.000
88-7303-341-5	Windows 95 For Dummies	L. 32.000
88-7303-345-8	Windows NT For Dummies Espresso	L. 19.000
88-7303-302-4	Windows NT Workstation 4.0 flash	L. 16.000
88-7303-157-9	Guida a Windows 95	L. 39.000
88-7303-138-2	Windows 95 Guida pratica	L. 28.000
88-7303-139-0	Windows 95 Grande guida	L. 65.000
88-7303-367-9	Office 97 For Dummies	L. 32.000
88-7303-328-8	Office 97 Autoistruzione	L. 42.000
88-7303-354-7	Office 97 per Windows For Dummies Espresso	L. 19.000
88-7303-323-7	Word 97 flash	L. 16.000
88-7303-342-3	Word 97 For Dummies	L.32.000
88-7303-347-4	Word 97 For Dummies Espresso	L. 19-000
88-7303-343-1	Excel 97 For Dummies	L. 32.000
88-7303-348-2	Excel 97 For Dummies Espresso	L. 19.000
88-7303-324-5	Excel 97 flash	L. 16.000
88-7303-339-3	I segreti di Excel 97	L. 78.000
88-7303-411-X	Laboratorio di Excel 97	L. 24.000
88-7303-174-9	Excel 95 flash	L. 16.000
88-7303-337-7	Excel 95 per tutti	L. 28.000
88-7303-169-2	Word 95 flash	L. 16.000
88-7303-336-9	Word 95 per tutti	L. 28.000
88-7303-199-4	PowerPoint 95 flash	L. 16.000
88-7303-178-1	Access 95 flash	L. 16.000
88-7303-170-6	I segreti di Word per Windows 95	L. 78.000

88-7303-161-7 I segreti di Excel per Windows 95 .... L. 78.000
88-7303-349-0 Access 97 For Dummies Espresso .... L. 19.000
88-7303-344-X Access 97 For Dummies .... L. 32.000
88-7303-412-8 Laboratorio di Access 97 .... L. 28.000
88-7303-325-3 Access 97 flash .... L. 16.000
88-7303-184-6 Access 95 per tutti .... L. 28.000
88-7303-163-3 Works per Windows 95 .... L. 49.000
88-7303-326-1 PowerPoint 97 flash .... L. 16.000
88-7303-198-6 Lotus Notes 4 flash ....^.......... L. 16.000
88-7303-353-9 Outlook 97 For Dummies Espresso .... L. 19.000
88-7303-449-7 Outlook 98 flash .... L. 16.000
88-7303-095-5 Windows 3.1, Word 6, Excel 5, Access 2 .... L 48.000
88-7303-038-6 Windows 3.1 Autoistruzione .... L. 48.000
88-7303-048-3 Windows per tutti quelli che .... L. 25.000
S8-7303-066-1 Winword per tutti (per Windows 3-1) .... L. 28.000
38-7303-078-5 Word 6 Autoistruzione ....^^.......... L. 35.000
38-7303-041-6 Il wordprocessing per tutti .... L. 25.000
38-7303-107-2 dBase 5 per Windows .... L. 28.000
38-7303-086-6 Access 2 Grande guida .... L 48.000
38-7303-037-8 Paradox per Windows Grande guida .... L. 85.000
38-7303-083-1 Excel 5 Autoistruzione .... L. 35.000
38-7303-161-7 I segreti di Excel per Windows 95 .... L. 78.000
38-7303-327-X Outlook 97 flash .... L. 16.000

38-7303-407-1 InterNet per tutti Terza edizione .... L. 32.000
38-7303-439-X Internet For Dummies Quinta edizione ....…../.......... L. 36.000
38-7303-356-3 Internet For Dummies Espresso ....(.......... L. 19.000
38-7303-422-5 Internet e Web flash .... L. 16.000
38-7303-152-8 Il mio server Web .... L. 65.000
38-7303-147-1 I segreti del World Wide Web .... L. 78.000
38-7303-140-4 Eudora, la posta elettronica via Internet .... L. 28.000
38-7303-314-8 Eudora flash .... L. 16.000
38-7303-206-0 Il modem per tutti .... L. 28.000
38-7303-120-X Internet per le aziende .... L. 48.000
38-7303-202-8 Intranet flash .... L. 16.000
38-7303-210-9 Telefonare con Internet .... L. 36.000
38-7303-151-X Il nuovo Navigare con Internet .... L. 58.000
«-7303-386-5 Internet Explorer 4 For Dummies .... L. 32.000
38-7303-383-0 Internet Explorer 4 For Dummies Espresso .... L. 19.000
58-7303-387-3 Netscape Communicator 4 For Dummies .... L. 32.000
38-7303-382-2 Netscape Communicator 4 For Dummies Espresso .... L. 19-000
Ì8-7303-408-X Internet Explorer 4 flash .... L. 16.000

88-7303-376-8 Netscape Communicator 4 flash .... L. 16.000
88-7303-319-9 Inglese per Internet flash .... L. 24.000
88-7303-477-2 Ricerche online For Dummies .... L. 36.000
88-7303-180-3 Netscape flash .... L. 16.000

88-7303-394-6 HTML 4 .... L. 48.000
88-7303-393-8 HTML 4 Tutto&Oltre .... L. 88.000
88-7303-409-8 FrontPage 98 Guida completa .... L. 59.000
88-7303-365-2 HTML 4 For Dummies .... L. 32.000
88-7303-355-5 HTML 4 For Dummies Espresso .... L. 19.000
88-7303-392-X HTML 4 flash .... L. 16.000
88-7303-363-6 Creare pagine Web For Dummies .... L. 32.000
88-7303-446-2 Costruire un sito Web For Dummies .... L. 39.500
88-7303-445-4 XML For Dummies .... L. 39.500
88-7303-188-9 PERL Guida pratica .... L. 36.000
88-7303-329-6 Guida a FrontPage 97 .... L. 32.000
88-7303-307-5 HTML 3.2 Guida completa .... L. 65.000
88-7303-196-X HTML flash .... L. 16.000
88-7303-320-2 HTML 3.2 Nuova edizione .... L. 38.000
88-7303-153-6 VRML .... L. 55.000
88-7303-352-0 Marimba Castanet la guida ufficiale .... L. 46.000
88-7303-391-1 HTML dinamico Guida completa .... L. 59.000

88-7303-305-9 Programmare in JavaScript .... L. 42.000
88-7303-316-4 Programmare in Visual J++ .... L. 58.000
88-7303-350-4 Java Restaurant .... L. 28.000
88-7303-357-1 JavaScript For Dummies Espresso .... L. 19.000
88-7303-423-3 Java For Dummies .... L. 39.500
88-7303-466-7 Java 1.2 Tutto&Oltre .... L. 88.000
88-7303-465-9 Java 1.2 Guida completa .... L. 69.000
88-7303-197-8 Java flash .... L. 16.000
88-7303-351-2 Java 1.1 Guida completa .... L. 69.000
88-7303-189-7 Java 1.0 Guida completa .... L. 64.000
88-7303-375-X Java 1.1 Tutto&Oltre .... L. 88.000

88-7303-410-1 AutoCAD 14 Guida completa .... L. 98.000
88-7303-360-1 AutoCAD 14 Guida all'uso .... L. 59-000
88-7303-414-4 AutoCAD 14 For Dummies... .... L. 32.000

ISBN	Titolo	Prezzo
88-7303-318-0	I segreti di AutoCAD 13	L. 78.000
88-7303-415-2	AutoCAD 14 For Dummies Espresso	L. 19-000
88-7303-044-0	AutoCAD 12 Comandi	L. 58.000
88-7303-400-4	3D Studio MAX 2 Guida completa	L. 69.000
88-7303-380-6	3D Studio MAX per il professionista	L. 48.000
88-7303-399-7	LightWave 3D 5.5 Guida completa	L. 88.000
88-7303-300-8	3D Studio MAX Guida completa	L.*65.000
88-7303-321-0	LightWave Guida completa	L. 56.000
88-7303-135-8	3D Studio 4	L. 58.000
88-7303-072-6	Guide Apogeo: 3D Studio 3	L. 48.000
88-7303-458-6	Photoshop 5 For Dummies	L. 36.000
88-7303-455-1	Photoshop 5 Guida completa	L. 69.000
88-7303-473-X	Photoshop 5	L. 98.000
88-7303-331-8	Photoshop 4 Guida completa	L. 59.000
88-7303-368-7	Photoshop 4 per Windows For Dummies	L. 32.000
88-7303-101-3	Photoshop 3 per Macintosh e Windows	L. 58.000
88-7303-068-8	Guide Apogeo: Photoshop 2.5	L. 28.000
88-7303-430-6	CorelDRAW! 8 For Dummies	L. 36.000
88-7303-440-3	CorelDRAW! 8 Guida completa	L. 59.000
88-7303-444-6	QuarkXPress Versione 4 per Mac e PC Guida all'uso	L. 49.000
88-7303-194-3	Lavorare con Adobe Illustrator	L 36.000
88-7303-089-0	CorelDRAW! 5	L. 68.000
88-7303-052-1	PageMaker 5	L. 53.000
88-7303-082-3	QuarXPress 3.1	L. 48.000
88-7303-154-4	Adobe Acrobat per Macintosh e Windows	L. 30.000
88-7303-060-2	Il desktop publishing per tutti quelli che	L. 25.000

ISBN	Titolo	Prezzo
88-7303-462-4	Visual Basic 6 Tutto&Oltre	L. 88.000
88-7303-456-X	Visual Basic 6 Guida completa	L. 69.000
88-7303-469-1	Visual Basic 6 For Dummies	L. 39-500
88-7303-304-0	Visual Basic flash	L. 16.000
88-7303-476-4	Visual Basic 6 For Dummies Espresso	L 19-000
88-7303-381-4	Visual Basic 5 For Dummies Espresso	L. 19-000
88-7303-463-2	I segreti di Visual Basic 6	L. 88.000
88-7303-470-5	Visual Basic - La programmazione dei database	L. 59-000
88-7303-402-0	A scuola di Visual Basic 5 For Dummies	L. 39-500
88-7303-338-5	Visual Basic 5 Guida completa	L. 69-000
88-7303-369-5	Visual Basic 5 For Dummies	L. 32.000
88-7303-155-2	Programmare in Visual Basic 4	L. 64.000
88-7303-164-1	Visual Basic 4 Guida pratica	L. 29-000
88-7303-098-X	SuperKit Visual Basic	L. 28.000
88-7303-071-8	Programare con Visual Basic 3	L. 58.000
88-7303-084-X	Borland C++ 4 e 4.5...	L. 58.000

88-7303-008-4 Programmare in C senza errori .... L. 23.000
88-7303-371-7 Borland C++ Builder La guida ufficiale .... L. 69.000
88-7303-333-4 I segreti di Visual Basic 5 .... L. 78.000
88-7303-454-3 Borland Delphi 4 Guida completa .... L. 88.000
88-7303-421-7 Borland C++ Builder 3 Guida completa .... L. 69.000
88-7303-474-8 Programmare con Delphi 4 .... L. 59.000
88-7303-361-X Programmare in C .... L. 64.000
88-7303-315-6 Programmare in C++ .... L. 59.000
88-7303-468-3 Visual C++ 6 For Dummies .... L. 39.500
88-7303-464-0 Visual C++ 6 Guida completa .... L. 69.000
88-7303-467-5 Visual C++ 6 Tutto&Oltre .... L. 88.000
88-7303-475-6 Visual C++ 6 For Dummies Espresso .... L. 19.000
88-7303-401-2 A scuola di C++ For Dummies .... L. 39.500
88-7303-370-9 Visual C++ 5 For Dummies .... L. 32.000
88-7303-374-1 Visual C++ 5 Guida completa .... L. 69.000
88-7303-306-7 Programmare in Visual C++ 4.2 .... L. 74.000
88-7303-372-5 Borland Delphi 3 La guida ufficiale .... L. 59.000
88-7303-201-X Delphi 2 .... L. 42.000

88-7303-397-0 MAC per tutti Nuova edizione .... L. 32.000
88-7303-384-9 Macintosh (MAC OS 8) For Dummies .... L. 32.000
88-7303-429-2 Computer in rete For Dummies .... L. 36.000
88-7303-204-4 Il computer per tutti Nuova edizione .... L. 28.000
88-7303-301-6 Computer flash .... L. 16.000
88-7303-200-1 Informatica Nuova edizione .... L. 58.000
88-7303-385-7 PC For Dummies Per DOS e Windows .... L. 32.000
88-7303-364-4 DOS For Dummies Per Windows 95 .... L. 32.000
88-7303-185-4 Il DOS per tutti .... L. 28.000
88-7303-418-7 UNIX For Dummies .... L. 36.000
88-7303-416-0 Linux For Dummies .... L. 39-500
88-7303-435-7 Red Hat Linux Tutto&Oltre .... L. 88.000
88-7303-419-5 UNIX For Dummies Espresso .... L. 19-000
88-7303-417-9 Linux For Dummies Espresso .... L. 19-000
88-7303-191-9 I segreti di Linux .... L. 69-000
88-7303-299-0 I segreti di UNIX .... L. 74.000
88-7303-317-2 Linux HowTo .... L. 65-000
88-7303-303-2 UNIX flash .... L. 16.000
88-85146-00-7 Lavorare con UNIX .... L. 58.000

88-7303-358-X Ingegneria economica del software .... L. 34.000
88-7303-4 78-0 Introduzione alla Matematica discreta .... L. 40.000

88-7303-413-6 Sistemica .... L. 24.000
88-7303-432-2 Atti del primo congresso italiano di Sistemica .... L. 60.000
88-7303-119-6 Vita artificiale .... L. 28.000
88-7303-002-5 Reti neuralì artificiali .... L. 25.000
88-7303-099-8 Intelligenza e vita .... L. 28.000

88-7303-441-1 Nozioni fondamentali delle reti .... L. 49.000
88-7303-443-8 NT Server 4 .... L.49.000
88-7303-457-8 NT Server 4 Enterprise .... L. 49.000
88-7303-472-1 Windows NT Workstation .... L. 49.000
88-7303-442-X TCP/IP .... L 49.000

88-7303-172-2 Telematica per la pace .... L. 26.000
88-7303-208-7 Sesto potere .... L. 32.000
88-7303-183-8 Il telefonino .... L. 24.000
88-7303-332-6 Incontri virtuali .... L. 24.000
88-7303-207-9 La vita sullo schermo .... L. 46.000
88-7303-359-8 Spaghetti hacker .... L. 30.000
88-7303-395-4 Internet per la didattica .... L 34.000
88-73\)3-448-9 Internet per chi studia nuova edizione .... L. 28.000
88-7303-453-5 Computer e scuola .... L. 24.000
88-7303-335-0 Fare marketing con Internet .... L. 28.000
88-7303-452-7 Trovare lavoro con Internet .... L. 24.000
88-7303-396-2 Gens electrica .... L. 28.000
88-7303-431-4 Ricerche blbliografiche in internet .... L. 24.000

88-7303-378-4 Cucina facile For Dummies .... L. 32.000
88-7303-379-2 Cucina dietetica For Dummies .... L. 32.000
88-7303-390-3 Tenersi in forma For Dummies .... L. 32.000
88-7303-434-9 Il mio cane For Dummies .... L. 32.000
88-7303-433-0 Il mio gatto For Dummies .... L. 32.000
88-7303-388-1 Musica classica For Dummies .... L. 39-500
88-7303-389-X Opera lirica For Dummies .... L. 39-500

Visual Basic 6, uno degli strumenti più potenti per creare rapidamente applicazioni sofisticate, richiede una guida approfondita, che non si fermi alle apparenze ma sia in grado di svelarne i segreti più nascosti. *I segreti di Visual Basic 6,* con centinaia di trucchi e suggerimenti utili, consente di impadronirsi delle tecniche di programmazione con Visual Basic 6 e di sfruttare appieno le nuove funzionalità dedicate a Internet e all'accesso ai dati. Con l'aiuto della guida esperta di Harold Davis e le numerose tecniche di codifica presentate nel volume, imparerete come creare applicazioni per Internet, oggetti dati e componenti ActiveX, aggiunte e creazioni guidate file di help e programmi di installazione.

- Help Composer e RoboHTML della Blue Sky Software
- SpyWorks, StorageTools, VersionTo e ActiveX Gallimaufry della Desawa
- InstallShield Express
- Sax mPower 98, Sax Basic Engine, Comm Objects e Sax Setup Wizard della Sax Software
- VS-Ocx 6.0, VSView 3.0, VSFlex 3.0 VSReports, VSData, VSDirect, VSSp e VSDocX della VideoSoft

L'autore:

Harold Davis è consulente senior nel gruppo Web Solutions della InForm1x Software, Inc., dove è specializzato nello sviluppo di soluzioni Web a livello enterprise per le aziende più importanti del mercato americano. È autore di numerosi bestseller su Visual Basic e sullo sviluppo per il Web.

Contenuti del libro

- Esplorare i nuovi strumenti dedicati al Web e le funzionalità Internet di Visual Basic.
- Estendere le funzioni di Visual Basic per creare applicazioni W complete in HTML dinamico.
- Sfruttare i nuovi strumenti client/server e l'accesso ai dati migliorato di Visual Basic.
- Reperire suggerimenti utili per creare applicazioni e controlli ActiveX di qualità superiore.
- Conoscere tecniche avanzate per lo sviluppo di database.
- Apprendere i segreti della creazione di programmi di installazio e di help professionali.

www.apogeonline.com

L. 88.000

ISBN 88-7303-463-2